Titre: Title:	Comparaison d'une méthode de génération de colonnes et d'une méthode de recherche tabou pour le problème d'horaires de véhicules avec dépôts multiples
Auteur: Author:	Guillaume Dereu
Date:	2005
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Dereu, G. (2005). Comparaison d'une méthode de génération de colonnes et d'une méthode de recherche tabou pour le problème d'horaires de véhicules avec dépôts multiples [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7363/

UNIVERSITÉ DE MONTRÉAL

COMPARAISON D'UNE MÉTHODE DE GÉNÉRATION DE COLONNES ET D'UNE MÉTHODE DE RECHERCHE TABOU POUR LE PROBLÈME D'HORAIRES DE VÉHICULES AVEC DÉPÔTS MULTIPLES

GUILLAUME DEREU
DÉPARTEMENT DE MATHÉMATIQUES ET DE GÉNIE INDUSTRIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(MATHÉMATIQUES APPLIQUÉES)
FÉVRIER 2005

Your file Votre référence
ISBN: 0-494-01305-2
Our file Notre référence
ISBN: 0-494-01305-2

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

COMPARAISON D'UNE MÉTHODE DE GÉNÉRATION DE COLONNES ET D'UNE MÉTHODE DE RECHERCHE TABOU POUR LE PROBLÈME D'HORAIRES DE VÉHICULES AVEC DÉPÔTS MULTIPLES

présenté par : DEREU Guillaume

en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de :

M. ROUSSEAU Louis-Martin, Ph.D., président

M. DESAULNIERS Guy, Ph.D., membre et directeur de recherche

M. HERTZ Alain, Doct. ès Sc., membre et codirecteur de recherche

M. CORDEAU Jean-François, Ph.D., membre

REMERCIEMENTS

Je tiens à remercier mon directeur de recherche, monsieur Guy Desaulniers, et mon co-directeur, monsieur Alain Hertz qui ont encadré mon travail pendant cette maîtrise. Ils ont été d'une grande disponibilité et ont su me prodiguer de bons conseils.

Je remercie également monsieur Jean-François Cordeau pour m'avoir autorisé à utiliser le code informatique de sa méthode de recherche tabou unifiée et aussi pour sa présence au sein du jury qui évaluera ce mémoire.

J'adresse enfin mes remerciements à monsieur Louis-Martin Rousseau qui a accepté d'être le président du jury d'examen de ce mémoire.

RÉSUMÉ

Ce mémoire porte sur le problème d'horaires de véhicules avec plusieurs dépôts (MDVSP). Ce problème consiste à effectuer au moindre coût un certain nombre de tâches avec des véhicules issus de plusieurs dépôts. Dans notre cas, chaque tâche devra être effectuée à un moment précis. Le coût se décompose quant à lui en un coût fixe pour chaque véhicule utilisé et en un coût proportionnel à la distance totale parcourue par l'ensemble des véhicules. Le but du mémoire est de comparer et de combiner la méthode de génération de colonnes et une méthode de recherche tabou pour résoudre ce problème. Dans le premier chapitre, on présente le cadre du problème. On décrit le problème et la façon dont sont générées les instances. On effectue, de plus, une revue de littérature et une brève description des contributions.

Dans le deuxième chapitre, on utilise de façon heuristique une méthode de génération de colonnes pour résoudre les problèmes étudiés. On commence par décrire la méthode de génération de colonnes pour ensuite présenter le modèle de génération de colonnes du MDVSP. Pour optimiser l'utilisation de la méthode de génération de colonnes, il faudra choisir l'algorithme avec lequel on résout le problème maître, le nombre de colonnes conservées et la stratégie de branchement. On envisagera également un arrêt prématuré de la méthode de génération de colonnes. Cette méthode nous fournit des solutions dont elle assure un saut d'optimalité d'environ 0,01 %. À la fin de ce chapitre, on appliquera un modèle de réoptimisation partielle aux solutions précédemment obtenues dans le but de les améliorer.

Le chapitre 3 décrit la méthode tabou utilisée pour résoudre les instances étudiées. On présente ensuite les adaptations rendues nécessaires par les premiers résultats obtenus. On utilisera cette méthode pour post-optimiser les solutions obtenues au

chapitre précédent. Cette approche s'est révélée décevante pour les tailles des problèmes considérés, ne fournissant que des solutions dont la qualité est très éloignée de celles obtenues avec l'approche de génération de colonnes.

Dans le dernier chapitre, nous avons développé un algorithme combinant métaheuristique et méthode de génération de colonnes qui produit régulièrement des solutions réalisables dont le coût diminue. Il est inspiré de la réoptimisation partielle effectuée au chapitre 2. Le principe de base est l'utilisation de la méthode de génération de colonnes pour réoptimiser des sous-ensembles de tâches. Cette dernière approche fournit des résultats de qualité supérieure à ceux obtenus avec la recherche tabou du chapitre 3.

ABSTRACT

This master thesis concerns the multiple depot vehicle scheduling problem (MDVSP). This problem consists of covering a set of tasks with vehicles from several depots at minimum cost. In our case, each task has a precise starting time. The cost is the sum of a fixed cost for each vehicle used and a cost proportional to the total distance traveled. The purpose of this master thesis is to compare and combine the column generation algorithm with a tabu search to solve this problem. In the first chapter, we present the framework of the problem. We describe the problem and the instances generator. Moreover, we provide a literature review and a short description of the contributions.

In the second chapter, we use a heuristic column generation algorithm to solve the problems. We describe this algorithm and then the mathematical formulation of the MDVSP. In order to optimize the use of column generation method, we have to choose the algorithm to solve the master problem, the number of columns and the branching strategy. We also consider a premature stop of the column generation. This method provides solutions with optimality gaps of approximately 0,01 %. At the end of the chapter, we apply a partial reoptimization to improve the previous solutions.

Chapter 3 describes the tabu search used to solve the problems and then some adaptations. We also used this method to post-optimize the solutions obtained in the preceding chapter. This approach turns out to be disappointing, providing solutions with bad quality in comparison with the column generation approach for the problem sizes considered.

In the final chapter, we introduce an algorithm that combines a meta-heuristic with a column generation that regularly produces feasible solutions whose cost decreases.

This algorithm is inspired by the partial reoptimization procedure described at the end of chapter 2. The basis for this method is the use of the column generation method to reoptimize subsets of tasks. This approach provides solutions with higher quality than those obtained with the tabu search of chapter 3.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ALGORITHMES

INTRODUCTION

Le problème d'horaires de véhicules avec plusieurs dépôts est un des nombreux problèmes dérivés du problème de tournées de véhicules (en anglais Vehicle Routing Problem). Ce dernier problème est défini depuis plus de 40 ans et est un des problèmes d'optimisation combinatoire les plus étudiés. Ils sont tous les deux NP-difficiles, ce qui signifie que le temps nécessaire pour les résoudre augmente exponentiellement avec la taille du problème. L'intérêt pour le problème de tournées de véhicules reste vif, Vigo et Toth ont ainsi édité un recueil d'articles sur le sujet en 2002.

Ces problèmes consistent à servir des clients ou des villes au moindre coût (ce qui est bien souvent équivalent à parcourir le moins de distance). Les applications sont très nombreuses. La plus évidente concerne les entreprises qui doivent effectuer de nombreuses livraisons à des endroits différents. Une autre application très importante est l'affectation des éléments d'une flotte de véhicules à des trajets, cela concerne les entreprises de transport en commun et les compagnies aériennes. Dans cette application, on parle plutôt de problèmes d'horaires de véhicules puisque, bien souvent, l'horaire des trajets est fixé.

Dans ce mémoire, on cherche à comparer les performances d'une méthode de génération de colonnes heuristique et une méthode de recherche tabou sur un problème d'horaires de véhicules avec plusieurs dépôts. L'étude de la méthode de génération de colonnes sur ce problème n'est pas nouvelle, mais il faudra néanmoins ajuster certains paramètres pour l'utiliser de la façon la plus adéquate et ainsi obtenir la meilleure base de comparaison possible. On choisira ensuite un algorithme de recherche tabou qui a donné de bons résultats sur des problèmes similaires à celui étudié. On lui appliquera quelques modifications pour l'adapter au problème étudié. À la suite de l'étude

de la méthode de génération de colonnes, on étudiera un modèle de réoptimisation locale. Ce dernier modèle et les résultats décevants de la recherche tabou pour les tailles des problèmes considérés nous ont conduits à introduire une nouvelle méthode méta-heuristique faisant appel à la génération de colonnes proposée.

On présentera le cadre de l'étude au premier chapitre, en particulier la description du problème et des instances ainsi qu'une revue de littérature. On utilisera de façon heuristique une méthode de génération de colonnes pour résoudre le problème au chapitre 2. Dans le chapitre 3, on utilisera un algorithme basé sur une recherche tabou pour résoudre le même problème. Le but est d'utiliser une méthode qui fournit des solutions régulièrement au cours de la résolution ce qui n'est pas le cas avec la méthode de génération de colonnes. Dans le dernier chapitre, on a développé un algorithme combinant méta-heuristique et méthode de génération de colonnes qui produit régulièrement des solutions réalisables dont le coût diminue.

CHAPITRE 1 : DESCRIPTION DU CADRE DE L'ÉTUDE

Dans ce premier chapitre, nous présentons d'abord le problème d'horaires de véhicules avec plusieurs dépôts (multiple depot vehicle scheduling problem ou MDVSP). Nous faisons également une revue de littérature et une brève description des contributions. Le générateur d'instances utilisé est décrit dans la dernière section.

1.1 Description du problème

On considère un ensemble de n tâches $N = \{1, 2, .., n\}$ que l'on doit couvrir où chaque tâche doit commencer précisement au temps a_i et a une durée de d_i. Soit t_{ij} le temps de trajet entre les tâches i et j. La tâche j peut succéder à la tâche i si $a_i+d_i+t_{ij} \leq a_j$. En effet, il n'est pas nécessaire d'arriver juste au temps a_j pour effectuer la tâche j, on peut arriver avant et attendre jusqu'à a_j. On considère également un ensemble de m dépôts $M = \{D_1, D_2, .., D_m\}$. Une tournée valide commence et finit au même dépot et est composée d'une suite de tâches qui peuvent se succéder. Une tournée correspond aux tâches qu'effectue un véhicule. On cherche une solution d'affectation des tâches dans des tournées de telle sorte que chaque tâche appartienne à une tournée et que l'on utilise au maximum v_k véhicules pour chaque dépot k. On cherche à minimiser le coût total qui se compose d'un coût fixe c_{fixe} par véhicule utilisé et d'un coût c_{ij} sur chaque succession de deux tâches i et j que l'on trouve dans les tournées choisies.

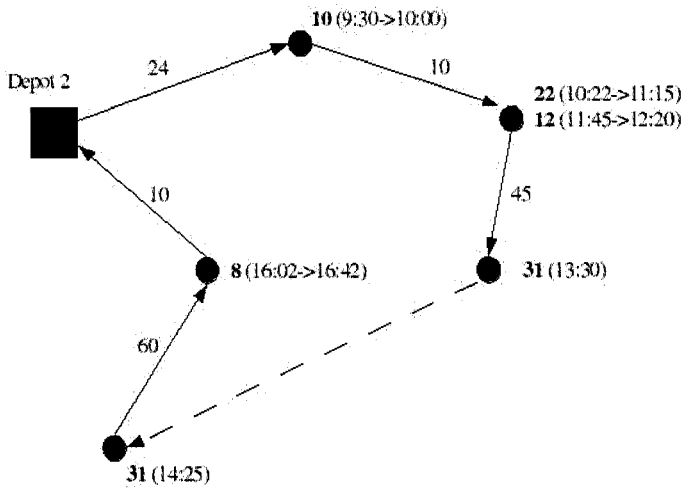

Figure 1.1 – Schéma d'une tournée valide

Pour illustrer le problème, une tournée valide est représentée à la figure 1.1. Partant du dépôt 2, le véhicule concerné se rend en 24 minutes à la tâche 10 qui commence à 9h30 et dure 30 minutes. Ensuite, il effectue les tâches 22, 12, 31 et 8. On peut remarquer que les tâches 22 et 12 ont la même localisation. On peut aussi noter la présence d'une tâche ne se terminant pas à l'endroit où elle a commencé (la tâche 31).

1.2 Revue de littérature

Le problème d'horaires de véhicules avec plusieurs dépôts a été étudié depuis plus de vingt ans. Bertossi, Cararessi et Gallo (1987) ont montré qu'il est NP-difficile pour

$m \geq 2$.

1.2.1 Méthode de génération de colonnes

Une méthode heuristique a été proposée par Dell'Amico, Fischetti et Toth (1993). Des méthodes exactes ont été proposées par Carpaneto et al. (1989) ; Ribeiro et Soumis (1994) ; Forbes, Holt et Watts (1994) ; Bianco, Mingozzi et Ricciardelli (1994) ; Löbel (1998) et Hadjar, Marcotte et Soumis (2001). Dans ce qui suit, nous discutons des approches exactes les plus pertinentes pour ce mémoire.

Ribeiro et Soumis (1994) ont proposé une approche par génération de colonnes imbriquée dans une méthode de séparation et évaluation progressive. La méthode de génération de colonnes permet de résoudre la relaxation linéaire du problème. Cette approche permet d'obtenir une borne sur la valeur de l'objectif beaucoup plus serrée que celles fournies par les méthodes utilisées précédemment. Les colonnes correspondant à des tournées de véhicules sont générées par un algorithme de plus court chemin, soit celui ayant le plus petit coût réduit. Les décisions de branchement sont prises sur la succession ou non de deux tâches. Cette approche leur permet de résoudre des instances possédant jusqu'à 300 tâches et 6 dépôts. La modélisation de Ribeiro et Soumis est utilisée dans la suite de ce mémoire et est décrite de façon détaillée à la section 2.2.

Hadjar, Marcotte et Soumis (2001) ont ajouté à cette approche la possibilité de fixer à 0 des variables dont les coûts réduits sont élevés. Ils introduisent également des plans coupants qui éliminent des solutions fractionnaires. Ceci permet d'améliorer la borne fournie par la relaxation linéaire du problème. De plus, ils utilisent deux stratégies de branchement : partitionnement des dépôts et fixation des inter-tâches. Le partitionnement des dépôts consiste à faire deux copies d'une tâche. Chacune est

uniquement accessible par une des deux partitions des dépôts. Une seule des deux copies doit être servie. La fixation des inter-tâches est la méthode utilisée précédemment par Ribeiro et Soumis (1994). Hadjar, Marcotte et Soumis donnent des résultats pour des instances avec 7 dépôts et de 350 à 2100 tâches.

Löbel (1998) utilise également une approche par génération de colonnes mais chaque colonne correspond à une variable X_{ij} qui vaut 1 si la tâche j succède à la tâche i, 0 sinon. La méthode est basé sur deux relaxations lagrangiennes différentes. Elle permet d'activer des chemins complets et pas seulement des arcs isolés. La méthode de génération de colonnes active, habituellement, les variables qui possèdent des coûts réduits négatifs. Ces variables promettent, en effet, une amélioration locale de la solution mais on ne prend pas en compte leurs interactions avec les autres variables qui ne sont pas actives. Löbel essaie de déterminer les variables non actives qui vont permettre une meilleure amélioration de la valeur de l'objectif. Chaque variable non active en lien avec certains voyages à vide des tournées de véhicule sélectionnées dans les relaxations lagrangiennes est susceptible d'être activée. Löbel résout des instances possédant jusqu'à 25000 tâches mais celles-ci ont des caractéristiques assez éloignées de celles étudiées dans ce mémoire. Il n'est donc pas possible d'effectuer une comparaison entre les deux approches.

Une extension naturelle du problème étudié est le problème d'horaires de véhicules avec plusieurs dépôts et fenêtres de temps (en anglais multi-depot vehicle scheduling problem with time windows ou MDVSPTW). La seule différence est que les tâches ne doivent plus nécessairement commencer à un moment précis mais leur début doit néanmoins appartenir à un intervalle. Ce problème est également NP-difficile car le MDVSP est un cas particulier de celui-ci. Il a été étudié par Bianco, Mingozzi et Ricciardelli (1995) qui résolvent des instances possédant jusqu'à 120 tâches et 5 dépôts. Ils généralisent la méthode publiée par les mêmes auteurs en 1994. Desaulniers, Lavigne et Soumis (1998) utilisent une méthode de génération de colonnes imbriquée dans un algorithme de séparation et évaluation progressive. Ils présentent une

version exacte et une version heuristique de leur algorithme, cette dernière leur permettant de résoudre des instances de 5 dépôts et 600 tâches. Cette approche est une généralisation de celle de Ribeiro et Soumis (1994).

1.2.2 Métaheuristiques

À notre connaissance, il n'y a pas de publications présentant des méta-heuristiques pour résoudre le problème d'horaires de véhicules avec dépôts multiples. Nous allons donc présenter quelques méta-heuristiques récentes permettant de résoudre le problème de tournées de véhicules (i.e., lorsque l'horaire des tâches n'est pas donné). Cette revue est inspirée de celle réalisée par Cordeau et al. (2004). Nous commençons par présenter quatre méthodes basées sur une recherche tabou, puis deux méthodes évolutives, un algorithme génétique et enfin un algorithme de type colonie de fourmis.

Cordeau, Gendreau et Laporte (1997) ont introduit un algorithme de recherche tabou permettant de résoudre des problèmes de tournées de véhicules avec plusieurs dépôts et de tournées de véhicules périodiques. Une version plus récente (2001) permet de résoudre des instances avec fenêtres de temps. C'est sur cet algorithme que l'on se basera dans le chapitre 3 de ce mémoire. Celui-ci sera donc décrit en détail au début de ce même chapitre.

Toth et Vigo (2003) utilisent une recherche tabou granulaire. Pour réduire le temps de calcul, ils suppriment l'arc entre deux clients lorsque ceux-ci sont distants de plus de β. Le nombre d'arcs est ainsi diminué de 10% à 20%. En pratique, ils font varier la valeur de β au cours de la résolution. Pour intensifier la recherche, ils diminuent la valeur de β et pour la diversifier ils augmentent β. Les solutions voisines de la solution courante sont obtenues en échangeant des arcs au sein d'une même route ou entre deux routes différentes.

Li, Golden et Wasil (2005) utilisent une recherche de succès en succès et une liste de voisins de longueur variable. Ils conservent uniquement une proportion α des 40 arcs incidents à chaque client les plus courts. Ils commencent par générer trois solutions différentes à l'aide de l'algorithme de Clarke and Wright (1964). Le voisinage d'une solution est obtenu en effectuant des mouvements 2-opt. Ils autorisent la détérioration de la valeur de la solution courante tant que celle-ci ne s'éloigne pas de plus de 1% de la valeur de la meilleure solution connue.

Ergun, Orlin et Steele-Felman (2003) utilisent une recherche avec un voisinage de très grande taille. Les solutions voisines sont obtenues par des mouvements 2-opt, des échanges de clients entre routes et des insertions de clients dans des routes différentes. L'ordre dans lequel ces mouvements doivent être appliqués est déterminé en résolvant un problème de plus court chemin sur un graphe auxiliaire. Cette approche permet plusieurs mouvements par itération mais chacune d'elle nécessite un temps de calcul important.

Prins (2004) propose un algorithme de recherche évolutive qui combine des opérations de croisements et de mutations. Les améliorations sont obtenues en appliquant des recherches locales sur les solutions. Ces recherches sont arrêtées dès que le coût de la solution augmente.

Mester et Bräysy (2005) combinent une recherche locale guidée et des stratégies évolutives. On conserve un ensemble de solutions. Chaque solution générée par une procédure de mutation n'a qu'une seule solution mère, celle-ci est remplacée si la nouvelle solution a une valeur plus faible.

Berger et Barkaoui (2004) utilisent un algorithme génétique hybride. Ils entretiennent deux ensembles de solutions auxquels ils peuvent appliquer une migration, en échangeant les meilleurs éléments de chaque ensemble. Des croisements sont effectués par

la création d'une solution fille à partir de deux solutions parents. Les solutions ainsi obtenues sont améliorées par l'application d'une recherche locale à voisinage large.

Reiman, Doerner et Hartl (2005) ont développé un algorithme de type colonie de fourmis. Ils appliquent alternativement deux phases. La première consiste à créer un ensemble de solutions de bonne qualité que l'on améliore à l'aide de mouvements 2-opt jusqu'à atteindre le critère d'arrêt. Un mécanisme d'apprentissage permet d'améliorer la qualité des solutions produites. Pour savoir si deux clients doivent être combinés, ils utilisent une valeur d'attraction entre les deux. Dans la deuxième phase, on décompose la meilleure solution identifiée dans la première phase en plusieurs sous-problèmes. On applique ensuite la méthode de la première phase à chaque sous-problème.

Les méta-heuristiques qui ont été développées pour résoudre des problèmes de tournées de véhicules sont généralement testées sur des instances comportant au plus 1000 tâches. De plus, les méta-heuristiques sont généralement comparées entre elles ou évaluées par rapport à l'écart les séparant d'une borne inférieure. On ne trouve que rarement des comparaisons entre des méta-heuristiques et des méthodes exactes, la raison étant que les méta-heuristiques ne sont utiles que lorsque les méthodes exactes ne peuvent plus produire une solution optimale en un temps raisonnable. Notons à nouveau, qu'à notre connaissance, aucune méta-heuristique avant celle présentée dans ce mémoire, n'a été développée spécifiquement pour résoudre des instances du MDVSP.

1.3 Contributions

Les contributions de ce mémoire sont les suivantes. Nous proposons d'abord une méthodologie pour optimiser l'utilisation de la méthode de génération de colonnes

heuristique pour la résolution des instances considérées. Pour cela, l'évolution des valeurs des variables sur lesquelles on pourrait brancher au cours de la résolution est étudiée. On espère ainsi être en mesure de choisir la meilleure stratégie de branchement.

Nous appliquons ensuite un algorithme de recherche tabou à de gros problèmes d'horaires de véhicules avec plusieurs dépôts où chaque tâche doit être effectuée à un moment précis, ce qui n'a jamais été fait à notre connaissance. Nous avons pour cela adapté un algorithme de recherche tabou résolvant des problèmes proches. Les performances de cet algorithme sont comparées à celles de la méthode de génération de colonnes pour démontrer la supériorité de la méthode de génération de colonnes pour les tailles des instances considérées.

Nous introduisons enfin un nouvel algorithme qui résout nos problèmes en fournissant régulièrement des solutions réalisables. Cet algorithme, qui combine des règles heuristiques, une liste taboue et la génération de colonnes pourrait être adapté pour résoudre d'autres variantes du problème de tournées de véhicules.

1.4 Description des instances

Le problème qui nous intéresse plus particulièrement consiste à affecter des parcours à des autobus provenant de plusieurs dépôts. C'est un cas particulier du problème d'horaires de véhicules avec plusieurs dépôts. Le générateur d'instance utilisé provient de Carpaneto et al. (1989).

Il est important de signaler que le début et la fin d'une tâche (un parcours) ne se trouvent pas toujours au même endroit. Chaque tâche i possède donc deux localisations, une pour le départ (x_i^d, y_i^d) et une pour l'arrivée (x_i^f, y_i^f). La distance de la tâche

i vers la tâche j est donc $d_{ij} = \sqrt{(x_i^f - x_j^d)^2 + (y_i^f - y_j^d)^2}$. Soit c_{fixe} le coût fixe pour un véhicule, on lui donne la valeur 10000. À partir de cela, on peut définir le coût c_{ij} à ajouter si un véhicule effectue successivement i puis j avec i et j appartenant à l'union des ensembles des dépôts et des tâches :

$$c_{ij} = \begin{cases} 0 & \text{si } i \in M \text{ et } j \in M \\ c_{fixe}/2 + \lfloor 10 d_{ij} \rfloor & \text{si } i \in M \text{ ou } j \in M \\ \lfloor 10 d_{ij} \rfloor & \text{sinon.} \end{cases}$$

Le temps pour se rendre de la fin de la tâche i au début de la tâche j est $t_{ij} = \lfloor d_{ij} \rfloor$.

Pour générer une instance, on commence par déterminer le nombre de points de relève qui sont en fait le nombre de points distincts $r \hookrightarrow \mathcal{U}[\lceil n/3 \rceil, \lfloor n/2 \rfloor]$ ($\hookrightarrow \mathcal{U}[a,b]$ signifie suit une loi uniforme sur l'intervalle $[a,b]$) que l'on va considérer dans le problème. Cela se justifie par le fait que, dans un ensemble de lignes autobus, les départs et arrivées des lignes ne sont pas tous distincts. On affecte à chaque point de relève et à chaque dépôt un couple de coordonnées $(x,y) \hookrightarrow \mathcal{U}[0,60] \times \mathcal{U}[0,60]$. Pour chaque dépôt, le nombre de véhicules disponibles est $v_k \hookrightarrow \mathcal{U}[\lceil 3 + \frac{n}{3m} \rceil, \lfloor 3 + \frac{n}{2m} \rfloor]$. Les tâches se divisent en deux types de tâches A et B. Les tâches de type A sont 40% du total et celles de type B 60%. Les premières ne se terminent pas à l'endroit où elles ont commencé. A chaque tâche de type A, on attribue aléatoirement un point de relève pour le départ et un pour l'arrivée de la tâche. Elles représentent des parcours sur le trajet d'une ligne d'autobus. Les secondes se terminent à l'endroit où elles ont commencé et représentent une succession d'allers et retours sur une même ligne d'autobus. À chaque tâche de type B, on attribue aléatoirement un point de relève qui permet de localiser le départ et l'arrivée de la tâche. Les tâches de type A sont réparties entre l'heure de pointe du matin A_1 (15%), l'heure de pointe de fin d'après-midi A_3 (15%) et le reste de la journée A_2 (70%). Les heures de début des tâches et leurs durées selon leur type sont données dans le tableau 1.1.

Tableau 1.1 – Heures de début et durées des tâches dans la journée

type	heure de début	durée
A_1	$\hookrightarrow \mathcal{U}[7h00, 8h00]$	$\hookrightarrow 5 + \mathcal{U}[0, 35] + \lfloor d_{ii} \rfloor$
A_2	$\hookrightarrow \mathcal{U}[8h00, 17h00]$	$\hookrightarrow 5 + \mathcal{U}[0, 35] + \lfloor d_{ii} \rfloor$
A_3	$\hookrightarrow \mathcal{U}[17h00, 18h00]$	$\hookrightarrow 5 + \mathcal{U}[0, 35] + \lfloor d_{ii} \rfloor$
B	$\hookrightarrow \mathcal{U}[5h00, 20h00]$	$\hookrightarrow 180 + \mathcal{U}[0, 120]$

Dans la suite, on considérera des problèmes de deux tailles : les premiers avec 4 dépôts et 500 tâches et les seconds avec 4 dépôts et 1000 tâches.

CHAPITRE 2 : MÉTHODE DE GÉNÉRATION DE COLONNES

Dans ce chapitre, on utilisera une méthode de génération de colonnes imbriquée dans une procédure de séparation et évaluation progressive pour résoudre des instances du problème décrit au chapitre 1. On utilisera le logiciel GENCOL dans lequel est implantée de façon très efficace une telle méthode de génération de colonnes. Afin de résoudre des problèmes de grande taille, nous avons recours à une version heuristique de cette approche. En effet, lorsque l'on se trouve à un noeud de branchement, on prendra des décisions sur lesquelles on ne reviendra plus. Dans la section 2.1, on décrit la méthode de génération de colonnes de façon générique. Dans la section 2.2, on présente le modèle de génération de colonnes que l'on utilise spécifiquement pour le MDVSP. Dans la section suivante, on cherche à optimiser l'utilisation de cette méthode. Dans la dernière section, on utilise un modèle de réoptimisation partielle dans le but d'améliorer les solutions précédemment trouvées.

2.1 Méthode de génération de colonnes

Certains problèmes linéaires comportent un très grand nombre de variables ce qui peut poser 2 types de difficultés : un manque d'espace mémoire lors du stockage de ces variables et un temps déraisonnable d'énumération de ces variables. Pour pallier ces difficultés, on utilise une méthode itérative qui résout alternativement deux problèmes. Le premier (appelé problème maître restreint) est, en fait, une version restreinte du problème original qui contient un très petit nombre de variables. Le second (appelé sous-problème) permet d'identifier les variables à ajouter au premier

problème. La méthode s'arrête en un nombre fini d'itértions, soit lorsque le second problème ne permet plus d'identifier de variables permettant d'améliorer la solution du premier problème.

2.1.1 Principes de base

La méthode de génération de colonnes a été introduite par Dantzig et Wolfe (1960) et Gilmore et Gomory (1961). Elle permet de résoudre des programmes linéaires possédant un nombre élevé de variables que l'on peut générer en résolvant un problème auxiliaire appelé sous-problème. L'efficacité de la méthode est basée sur le fait que les sous-problèmes sont faciles à résoudre.

Le programme linéaire est généralement appelé problème maître (PM) et peut se formuler ainsi :

$$\text{(PM)} \quad \text{Minimiser} \quad \sum_{p\in\Omega} c_p\theta_p \tag{2.1}$$

sujet à

$$\sum_{p\in\Omega} a_{jp}\theta_p = b_j \quad \forall j \in J, \tag{2.2}$$

$$\theta_p \geq 0 \quad \forall p \in \Omega. \tag{2.3}$$

Ω est l'ensemble des indices p des variables θ_p. J est l'ensemble des contraintes avec a_{jp} les coefficients des variables θ_p et b_j les membres de droite. Les coefficients des variables θ_p dans l'objectif sont quant à eux notés c_p. On suppose qu'il y a beaucoup plus de variables que de contraintes (i.e. $\mid \Omega \mid \gg \mid J \mid = m$), ce qui est le cas le plus courant d'application de la méthode de génération de colonnes.

De plus, on reformule l'hypothèse qui impose la possibilité de générer les variables en résolvant un sous-problème (SP). Il est nécessaire qu'il existe un sous-problème qui, à chaque vecteur $\pi^T = (\pi_1, .., \pi_m)$ de variables duales associé aux contraintes (2.2), permet de trouver une colonne $a_q = \begin{pmatrix} a_{1q} \\ \vdots \\ a_{mq} \end{pmatrix}$ telle que $c_q - \pi^T a_q = \min_{p \in \Omega}\{c_p - \pi^T a_p\}$. En d'autres termes, le sous-problème permet de générer une colonne de coût réduit minimum. Pour le MDVSP, nous verrons à la section 2.2 que le sous-problème est un problème de plus court chemin.

2.1.2 Algorithme

L'algorithme de génération de colonnes est un processus itératif où, à chaque itération on résout tour à tour un problème appelé problème maître restreint et le sous-problème. À l'itération i, le problème maître restreint (PMR_i) coorespond au PM restreint à un petit sous-ensemble $\Omega_i \subset \Omega$ des variables :

$$(PMR_i) \quad \text{Minimiser} \quad \sum_{p \in \Omega_i} c_p \theta_p \tag{2.4}$$

sujet à

$$\sum_{p \in \Omega_i} a_{jp} \theta_p = b_j \quad \forall j \in J, \tag{2.5}$$

$$\theta_p \geq 0 \quad \forall p \in \Omega_i. \tag{2.6}$$

La résolution du PMR_i fournit une solution réalisable pour le PM et π_i le vecteur des variables duales associées aux contraintes (2.5).

Algorithme 2.1 Méthode de génération de colonnes

$i = 1$
Determiner Ω_1 tel que PMR_1 soit réalisable
Résoudre le PMR_1 avec Ω_1 (fournit une solution primale θ_1 et une sol. duale π_1)
Résoudre le SP_1 avec π_1 (fournit une solution x_1 de coût z_1^{SP})
Tant que $z_i^{SP} < 0$
- Créer une nouvelle variable θ_p en calculant c_p et a_{jp} à partir de x_i
- $\Omega_{i+1} = \Omega_i \cup \{p\}$
- $i = i + 1$
- Résoudre le PMR_i avec Ω_i (fournit une solution primale θ_i et une sol. duale π_i)
- Résoudre le SP_i avec π_i (fournit une solution x_i de coût z_i^{SP})

Les variables duales fournies par le PMR_i permettent de formuler le SP. Le SP fournit alors la variable θ_p ayant le plus petit coût réduit $c_p - \pi^T a_p$. Si celui-ci est positif, on est sûr que la solution est optimale car cela signifie qu'aucune variable n'appartenant pas à Ω_i ne permettrait de diminuer l'objectif. Sinon, la solution n'est pas optimale et alors on ajoute la variable θ_p à Ω_i pour former Ω_{i+1}. On peut commencer l'itération suivante en résolvant PMR_{i+1}. La méthode est décrite dans l'algorithme 2.1.

Pour en connaître davantage sur l'application de la méthode de génération de colonnes pour résoudre des problèmes de tournées de véhicules et d'horaires d'équipages, le lecteur peut consulter les articles synthèse de Desaulniers et al. (1998) et Barnhart et al. (1998).

2.2 Modèle de génération de colonnes pour le MDVSP

Dans la littérature, plusieurs formulations ont été proposées pour le MDVSP. Une qui s'est avérée efficace lorsque combinée à une méthode de génération de colonnes est celle proposée par Ribeiro et Soumis (1994). C'est cette formulation que nous utiliserons dans ce mémoire.

Soit Ω l'ensemble des tournées possibles. Pour $p \in \Omega$, on définit c_p le coût de la tournée p, a_{ip} une constante binaire qui prend la valeur 1 si et seulement si la tournée p couvre la tâche i, b_p^k une constante binaire qui prend la valeur 1 si et seulement si la tournée p commence au dépôt D_k et θ_p une variable binaire qui vaut 1 si et seulement si la tournée p est utilisée.

Si on reprend la tournée p de la figure 1.1, on obtient

$c_p = c_{fixe} + c_{D_2,10} + c_{10,22} + c_{22,12} + c_{12,31} + c_{31,8} + c_{8,D_2}$,

$$a_{ip} = \begin{cases} 1 \text{ si } i \in A = \{8, 10, 12, 22, 31\} \\ 0 \text{ sinon} \end{cases}, \quad b_p^k = \begin{cases} 1 \text{ si } k = D_2 \\ 0 \text{ sinon.} \end{cases}$$

Le MDVSP se formule alors :

$$\text{Minimiser} \qquad \sum_{p \in \Omega} c_p \theta_p \tag{2.7}$$

sujet à

$$\sum_{p \in \Omega} a_{ip} \theta_p = 1 \qquad \forall i \in N, \tag{2.8}$$

$$\sum_{p \in \Omega} b_p^k \theta_p \leq v_k \qquad \forall k \in M, \tag{2.9}$$

$$\theta_p \in \{0, 1\} \qquad \forall p \in \Omega. \tag{2.10}$$

La fonction objectif (2.7) cherche à minimiser le coût total. Les contraintes (2.8) s'assurent que chaque tâche soit couverte une et une seule fois. Les contraintes (2.9) s'assurent que le nombre de véhicules disponibles par dépôt ne soit pas dépassé.

Ce programme linéaire en nombres entiers est complet tel qu'il est présenté précédemment mais il est difficilement exploitable car il comporte un très grand nombre de variables (une par tournée). Ribeiro et Soumis (1994) ont alors proposé une approche de génération de colonnes imbriquée dans une méthode de séparation et d'évaluation

progressive. Dans une telle approche, on remplacera Ω par un de ces sous-ensembles Ω'. Le programme linéaire sera maintenant appelé problème maître (problème maître restreint pour la résolution sur Ω') et l'on générera les colonnes de Ω' à l'aide de sous-problèmes.

Soit α_i $(i \in N)$ et β_k $(k \in M)$ les variables duales du problème maître restreint associées aux contraintes (2.8) et (2.9) à une itération donnée. Pour chaque dépôt k, on peut alors générer la colonne (i.e. la tournée) de coût marginal minimum en résolvant le sous-problème k suivant.

Le sous-problème k est un problème de plus court chemin dont le réseau $G_k = (V_k, A_k)$ est composé de deux noeuds pour le dépôt (un comme noeud origine D_k et un comme noeud destination D'_k) et d'un noeud par tâche ($V_k = D_k \cup D'_k \cup N$). L'ensemble des arcs comprend tous les arcs de la forme (D_k, i) avec un coût $c_{D_k i} - \beta_k$ et (i, D'_k) avec un coût $c_{iD_k} - \alpha_i$ pour tout $i \in N$. Il comprend également les arcs (i, j) si la tâche j peut succéder à la tâche i avec un coût $c_{ij} - \alpha_i$. Sur ce réseau, on cherche le plus court chemin pour se rendre de D_k à D'_k. Le réseau associé à un sous-problème k est représenté à la figure 2.1. Il correspond à une instance avec 4 tâches. Pour plus de commodité, on a numéroté les tâches dans l'ordre croissant des heures de début. Dans notre exemple, il n'y a pas d'arc entre 3 et 4, on peut imaginer que celles-ci sont éloignées ou que la tâche 3 dure longtemps.

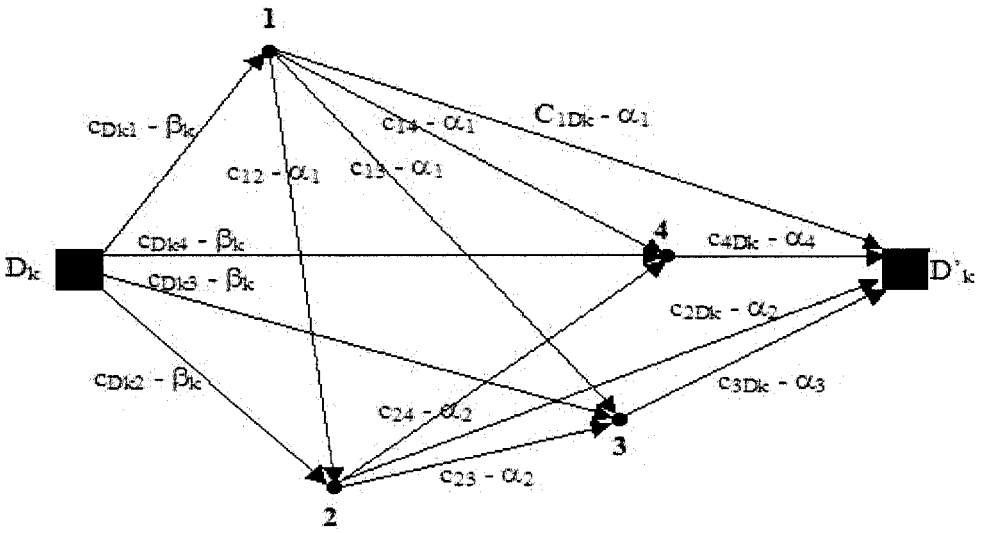

Figure 2.1 – Réseau du sous-problème k

2.3 Stratégies d'accélération / Résultats

Pour les problèmes de grande taille, on utilise une approche heuristique de génération de colonnes. Nous allons essayé d'utiliser au mieux cette approche, c'est-à-dire l'accélérer au maximum en détériorant au minimum la qualité des solutions produites. On procède, pour cela, en plusieurs étapes. On commence par accélerer la résolution de la relaxation linéaire (section 2.3.1). Ensuite, on teste quelques stratégies de branchement simples et mixtes (section 2.3.2). Pour finir, on tente d'arrêter prématurément la méthode de génération de colonnes et de limiter le nombre de décisions de branchement simultanées (section 2.3.3).

2.3.1 Accélération de la résolution de la relaxation linéaire

Pour accélerer la résolution de la relaxation linéaire, on peut modifier plusieurs paramètres. Il faut choisir en premier lieu l'algorithme que l'on utilise dans GENCOL pour résoudre les PMR_i : on a le choix entre l'algorithme du simplexe primal, l'algorithme du simplexe dual et la méthode de barrière. Deux autres paramètres importants lors de la résolution par génération de colonnes sont les nombres de colonnes minimum et maximum que l'on s'autorise à conserver dans le problème maître. Pour être certain d'avoir le nombre de colonnes entre les deux bornes désirées, on commence par générer des colonnes pour en obtenir le nombre minimum recherché. Ensuite, on en génère jusqu'à arriver au maximum et là on supprime les colonnes avec les coûts réduits les plus grands pour redescendre au nombre minimum de colonnes et on recommence. Si l'on conserve trop de colonnes, le problème maître va devenir très gros et nécessitera donc de longs temps de résolution. De plus, si l'on conserve trop peu de colonnes dans le problème maître, on risque d'avoir à générer un grand nombre de fois les mêmes colonnes, ce qui peut également se révéler une perte de temps considérable. Les deux

paramètres qui vont nous permettre de modifier les nombres minimum et maximum de colonnes sont $\overline{m}$ qui, multiplié par le nombre de contraintes, donne le maximum de colonnes que l'on conserve dans le problème maitre restreint et $\underline{m}$ qui, multiplié par le nombres de contraintes, donne le minimum de colonnes que l'on conserve dans le problème maitre.

Des tests ont été effectués sur 10 instances de chaque catégorie de problèmes. Les résultats sont rapportés dans les tableaux 2.1 (problèmes de 500 tâches et 4 dépôts) et 2.2 (problèmes de 1000 tâches et 4 dépôts). Les résultats fournis sont les moyennes des temps CPU et le nombre d'itérations de génération de colonnes mis pour résoudre les instances en fonction des paramètres.

Tableau 2.1 – Relaxation linéaire : 4 dépôts et 500 tâches

		primal		dual		barrière	
$\underline{m}$	$\overline{m}$	temps	# itérations	temps	# itérations	temps	# itérations
1	2	708 s	720	988 s	693	a	
1.25	2	480 s	424	687 s	429	a	
1.5	3	**367 s**	259	520 s	266	b	
2.5	5	370 s	188	482 s	188	425 s	214
4	8	434 s	169	537 s	168	423 s	183
5	10	484 s	165	598 s	165	447 s	179

Tableau 2.2 – Relaxation linéaire : 4 dépôts et 1000 tâches

		primal		dual		barrière	
$\underline{m}$	$\overline{m}$	temps	# itérations	temps	# itérations	temps	# itérations
1.25	2	6866 s	1168	11084 s	1190	c	
1.5	3	5164 s	657	8058 s	667	d	
2.5	5	4978 s	417	7054 s	417	4803 s	464
4	8	5995 s	364	7978 s	361	**4731 s**	380
5	10	6371 s	349	7805 s	347	4789 s	364

Dans le tableau 2.1, trois résultats n'ont pu être obtenus. Pour les deux premiers (noté a dans le tableau), la première instance n'était toujours pas résolu après 50000 itérations. La configuration noté b a permis de résoudre les trois premières instances (en respectivement 1164 s, 1784 s et 46031 s) mais pas la quatrième malgré les 40000 itérations laissées avant l'arrêt.

Dans le tableau 2.2, deux résultats n'ont pu être obtenus. Dans la configuration noté c, la première instance n'a pu être résolue en moins de 5000 itérations. Cette instance a été résolue en 6246 secondes dans la configuration d, mais dans ce cas la seconde instance n'a pas été résolue lors des 5000 itérations accordées.

Pour tous les algorithmes utilisés et les problèmes résolus, le fait d'augmenter le nombre de colonnes (i.e. $\underline{m}$ et $\overline{m}$) permet de diminuer le nombre d'itérations mais celles-ci nécessitent plus de temps. Lorsque le nombre de colonnes est petit, l'effet prédominant est la diminution du nombre d'itérations. Ainsi, lorsque l'on augmente le nombre de colonnes le temps de résolution diminue. Cela n'est plus vrai pour un nombre plus élevé de colonnes. Dans ce cas, le nombre d'itérations ne diminue presque plus, alors que le temps par itération continue à augmenter. Alors, lorsque l'on augmente le nombre de colonnes, le temps de résolution augmente également. C'est pourquoi, pour chaque algorithme, on identifie le temps minimum de résolution pour une stratégie intermédiaire.

Pour les problèmes de 500 tâches (et 4 dépôts), on observe que c'est l'algorithme du simplexe primal avec $\underline{m}$=1.5 et $\overline{m}$ =3 qui se révèle le plus performant (en gras dans le tableau 2.1). Pour les problèmes de 1000 tâches (et 4 dépôts), on observe que c'est l'algorithme de barrière avec $\underline{m}$=4 et $\overline{m}$ =8 qui se révèle le plus performant (en gras dans le tableau 2.2). Pour la suite, ce sont donc ces paramètres que l'on va conserver lors de la résolution complète des problèmes.

2.3.2 Stratégies de branchement

Comme on l'a déjà précisé, on ne revient pas sur les décisions de branchement effectuées. C'est pourquoi il faut apporter un soin particulier à ces décisions qui peuvent affecter de façon considérable l'objectif final si elles sont prises de façon inappropriée. Des statégies de branchement sur les inter-tâches, sur les colonnes et une combinaison des deux ont été testées. Le but de cette section est de choisir la façon dont on va effectuer les branchements dans l'arbre de la figure 2.2. Dans cette section, nous présentons et discutons les résultats obtenus.

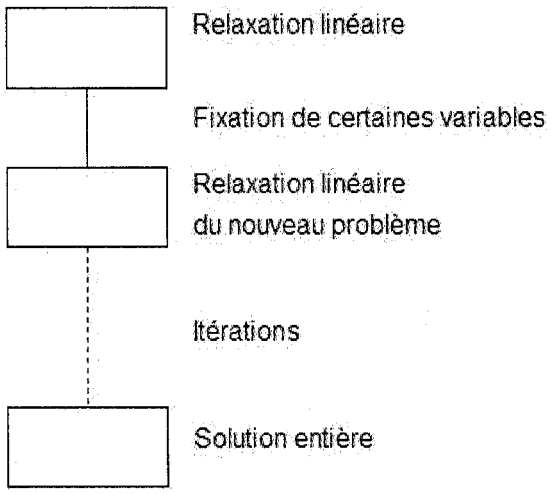

Figure 2.2 – Arbre de branchement

Stratégies de branchement sur les inter-tâches

On définit les variables d'inter-tâches it_{ij} avec i et j appartenant à l'union de l'ensemble des tâches et de l'ensemble des dépôts par la formule ci-dessous où Ω_{ij} est

l'ensemble des tournées telles qu'elles couvrent i et j et la tâche j succède immédiatement à la tâche i.

$$it_{ij} = \sum_{p \in \Omega_{ij}} \theta_p \tag{2.11}$$

Dans une solution réalisable, si la tâche j suit la tâche i alors cette variable (it_{ij}) vaut 1, si ce n'est pas le cas elle vaut 0. Néanmoins, dans le cas général, ces variables ne sont pas entières au cours de la résolution, elles sont plutôt restreintes à l'intervalle $[0, 1]$.

Les stratégies testées sont décrites ci-dessous :

- La stratégie 1 correspond à choisir une seule inter-tâche à chaque fois (le plus proche de 1) et de la fixer à 1.
- Les stratégies 0X fixent à 1 toutes les inter-tâches dont la valeur est supérieure à 0.X.

  S'il n'y a pas de candidat possible, on branche sur les colonnes.

Lorsque que l'on doit brancher sur les colonnes, on en choisit entre une et dix de sorte que leurs valeurs θ_p soient toutes supérieures à 0.7. On impose de plus que la somme des $1 - \theta_p$ pour les colonnes sélectionnées soit inférieure à 0.7.

Soit t0.X le taux de variables (1 par inter-tâche) qui sont plus grandes que 0.X à une itération et qui sont à 1 à la fin. Dans les problèmes de 500 tâches, il y a environ 600 inter-tâches dans la solution finale et, dans les problèmes de 1000 tâches, ce nombre s'élève à 1200.

Des tests ont été effectués sur 5 instances de chaque catégorie de problèmes. Les résultats sont rapportés dans les tableaux 2.3 (problèmes de 500 tâches et 4 dépôts) et 2.4 (problèmes de 1000 tâches et 4 dépôts). Les résultats fournis sont les moyennes des sauts, des temps CPU mis pour résoudre les instances, le nombre de noeuds de

l'arbre de branchement et les taux de branchement t0.5, t0.6, t0.7, t0.8 et t0.9 qui sont définis ci-dessus en fonction des paramètres. Le saut correspond à la l'écart relatif entre l'objectif de la solution obtenue (z) et l'objectif de la relaxation linéaire (z_{RL}).

$$Saut = \frac{z - z_{RL}}{z_{RL}} \tag{2.12}$$

Tableau 2.3 – Branchement sur les inter-tâches : 4 dépôts et 500 tâches

Stratégie	saut(%)	temps(s)	noeuds	t0.5	t0.6	t0.7	t0.8	t0.9
1	0.026	869.8	93.2	0.74	0.85	0.91	0.93	0.95
09	0.019	509.2	24.8	0.76	0.88	0.92	0.94	**0.95**
08	0.017	455.7	10.6	0.79	0.90	0.94	**0.96**	0.96
07	**0.016**	428.0	3.6	0.85	0.94	**0.98**	0.98	0.98
06	0.044	400.0	2.2	0.90	**0.98**	0.98	0.98	0.98
05	0.324	385.8	1.2	**0.92**	0.96	0.96	0.96	0.95

Tableau 2.4 – Branchement sur les inter-tâches : 4 dépôts et 1000 tâches

Stratégie	saut(%)	temps(s)	noeuds	t0.5	t0.6	t0.7	t0.8	t0.9
1	0.020	16619	243	0.77	0.84	0.89	0.92	0.94
09	0.016	7674	55	0.80	0.87	0.93	0.96	**0.97**
08	0.017	6611	26.8	0.81	0.90	0.94	**0.96**	0.96
07	**0.009**	5424	3.6	0.90	0.95	**0.98**	0.98	0.98
06	0.033	4975	1.6	0.94	**0.98**	0.98	0.98	0.98
05	0.088	4951	1.4	**0.97**	0.97	0.97	0.97	0.97

On remarque que plus la stratégie de branchement impose de décisions à chaque noeud, plus le temps de résolution et le nombre de noeuds sont faibles. Ceci est normal car plus on impose de décisions de branchement à un noeud plus on s'approche d'une solution entière.

L'évolution des sauts n'est pas aussi monotone, on remarque néanmoins que ceux-ci augmentent fortement pour les stratégies très contraignantes (05 et 06). Les sauts

sont plus faibles pour les gros problèmes (1000 tâches) que pour les problèmes moyens (500 tâches).

On note D la diagonale reliant la case (05,t0.5) en bas à gauche et la case (09,t0.9) en haut à droite. Les valeurs en dessous de la diagonale D ne sont pas tout à fait à 1 car certaines inter-tâches passent de 1 à 0. Elles sont donc comptabilisées comme étant supérieure à toutes les valeurs sans pour autant être à 1 à la fin. En effet, on ne branche pas sur les variables déjà entières. Au-dessus de la diagonale, on observe bien une croissance lorsque les ratios deviennent plus contraignants (de t0.5 à t0.9).

La meilleure stratégie semble être 07 ; elle offre un bon compromis saut / temps de résolution. On peut essayer d'expliquer cela par les écarts importants entre t0.5 et t0.6 de l'ordre de 8% et entre t0.6 et t0.7 de l'ordre de 5% par rapport à celui assez faible entre t0.7 et t0.8 de l'ordre de 2%.

Stratégies de branchement sur les colonnes

Une colonne correspond à une tournée. Par colonne, on désigne en fait la variable θ_p qui indique avec quel flot la tournée p est choisie. Dans une solution réalisable, si la tournée p est choisie $\theta_p = 1$ et $\theta_p = 0$ sinon. Néanmoins, dans le cas général, ces variables ne sont pas entières au cours de la résolution, elles sont plutôt restreintes à l'intervalle $[0, 1]$.

Les stratégies testées sont décrites ci-dessous :

- La stratégie 1 correspond à choisir une seule colonne à chaque fois (la plus proche de 1) et de la fixer à 1.
- Les stratégies 0X fixent à 1 toutes les colonnes dont la valeur est supérieure à 0.X. S'il n'y a pas de candidat possible, on branche sur la colonne la plus proche de 1.

Soit t0.X le taux de variables qui sont plus grandes que 0.X à une itération et qui sont à 1 à la fin. Dans les problèmes de 500 tâches, il y a environ 110 colonnes dans la solution finale et, dans les problèmes de 1000 tâches, ce nombre s'élève à 210.

Les résultats sont rapportés dans les tableaux 2.5 (problèmes de 500 tâches et 4 dépôts) et 2.6 (problèmes de 1000 tâches et 4 dépôts).

Tableau 2.5 – Branchement sur les colonnes : 4 dépôts et 500 tâches

Stratégie	saut(%)	temps(s)	noeuds	t0.5	t0.6	t0.7	t0.8	t0.9
1	0.015	622	48.6	0.44	0.66	0.78	0.83	0.84
09	0.017	530	33.6	0.44	0.66	0.77	0.84	**0.86**
08	0.019	489	24.4	0.42	0.71	0.82	**0.88**	0.82
07	**0.013**	427	8	0.52	0.84	**0.94**	0.92	0.89
06	0.024	397	3.6	0.60	**0.93**	0.91	0.88	0.85
051	0.048	393	3.8	0.64	**0.90**	0.88	0.84	0.81

Les remarques faites pour les branchements sur les inter-tâches sont toujours valables. La meilleure stratégie trouvée précédemment pour les problèmes de 500 tâches (branchement sur les inter-tâches plus grandes ou égales à 0.7) est détronée par celle qui consiste à brancher sur les colonnes ayant des valeurs plus grandes ou égales à 0.7. En effet, les sauts sont de 0.013% contre 0.016% pour un temps CPU comparable (428 s et 427 s). En revanche, pour les problèmes de 1000 tâches, la meilleure stratégie reste celle qui consiste à brancher sur les inter-tâches ayant des valeurs plus grandes ou égales à 0.7.

Tableau 2.6 – Branchement sur les colonnes ; 4 dépôts et 1000 tâches

Strategie	gapsaut(%)	temps(s)	noeuds	t0.5	t0.6	t0.7	t0.8	t0.9
1	**0.010**	9954	108.8	0.42	0.61	0.73	0.82	0.85
09	0.013	8114	84.8	0.47	0.65	0.78	0.84	**0.88**
08	0.014	6126	43.4	0.50	0.71	0.81	**0.89**	0.82
07	0.014	5804	13.8	0.57	0.85	**0.96**	0.96	0.94
06	0.027	4983	4.2	0.73	**0.97**	0.95	0.94	0.92
051	0.033	4788	3.6	0.91	**0.96**	0.95	0.94	0.91

Stratégie de branchement mixtes : colonnes ou inter-tâches

On va maintenant combiner les deux stratégies utilisées précédemment. Pour cela, on pondère les scores obtenus par chaque méthode de branchement pour pouvoir les comparer entre elles. Le score de la méthode de branchement sur les colonnes est donné par le plus petit flot parmi les colonnes sélectionnées. Pour trouver le score S de la méthode de branchement sur les inter-tâches, il faut appliquer successivement les trois formules suivantes. Cela revient à soustraire au score moyen l'écart type. La base du score est donc obtenue en calculant le flot moyen des inter-tâches sélectionnées. Un score élevé est donc associé à des flots élevés, ce qui est logique car dans ce cas les décisions de branchement sont peu risquées. De plus, on soustrait au score moyen l'écart type ce qui permet de favoriser les cas où les flots sont moins dispersés. Dans ce cas, les décisions sont plus cohérentes les unes avec les autres car de nombreuses variables d'inter-tâches choisies appartiennent aux mêmes tournées. Soit T l'ensemble des couples de tâches des inter-tâches sélectionnées.

$$S = \sum_{(i,j)\in T} \frac{it_{ij}}{\mid T \mid}, \tag{2.13}$$

$$S_2 = \sum_{(i,j)\in T} \frac{it_{ij}^2}{\mid T \mid} - S^2, \tag{2.14}$$

$$S = S - \sqrt{S_2}. \tag{2.15}$$

Soit X et Y les coefficients de pondération respectifs des méthodes de branchement sur les colonnes et sur les inter-tâches. Plus précisément, on compare les produits des scores des méthodes par les pondérations pour déterminer la méthode sélectionnée. Si $X \times score_X \leq Y \times score_Y$, on choisira la méthode Y (branchement sur les inter-tâches). Dans le cas contraire, on choisira la méthode X (branchement sur les colonnes). De plus, on autorise les branchements seulement sur les variables qui ont un flot supérieur à 0.7. Si aucun branchement n'est possible, on prendra la colonne avec le flot maximum. On prend la valeur 0.7 car c'est celle qui a le mieux fonctionné précédemment.

Les résultats sont rapportés dans les tableaux 2.7 (problèmes de 500 tâches et 4 dépôts) et 2.8 (problèmes de 1000 tâches et 4 dépôts).

Tableau 2.7 – Stratégie mixte : 4 dépôts et 500 tâches

X	Y	saut(%)	temps(s)	noeuds
0	1	0.0164	428	3.6
0.7	1	0.0164	417	3.6
0.8	1	0.0164	413	3.6
0.9	1	0.0164	414	3.6
1	1	0.016	415	4.4
1	0.9	0.0142	431	8.4
1	0.8	0.0136	422	7.4
1	0.7	**0.0132**	427	8
1	0	**0.0132**	427	8

Pour le tableau 2.7, on observe que toutes les stratégies pour X strictement inférieur à 1 et Y=1 sont identiques. En effet, les inter-tâches prennent toujours des valeurs supérieures à celles prises par les colonnes. Une inter-tâche peut être présente dans plusieurs colonnes. C'est pourquoi ces tests n'ont pas été reproduits dans le cas des problèmes avec 1000 tâches. Dans les deux cas, ces tests n'ont pas permis de trouver une stratégie qui permette de diminuer les sauts minimums obtenus.

Tableau 2.8 – Stratégie mixte : 4 dépôts et 1000 tâches

X	Y	saut(%)	temps(s)	noeuds
0	1	**0.009**	5424	3.6
1	1	**0.009**	5507	3.6
1	0.9	0.016	5700	15.2
1	0.8	0.0132	5581	16.2
1	0.7	0.0128	5436	15
1	0.6	0.014	5783	13.8
1	0.5	0.014	5753	13.8
1	0	0.014	5804	13.8

2.3.3 Arrêt prématuré

La courbe de la figure 2.3 présente l'évolution de la valeur de l'objectif en fonction de l'itération dans la méthode de génération de colonnes. Ceci illustre le fait que, pendant les dernières itérations, l'objectif n'évolue presque plus. Cela nous laisse envisager la possibilité de pouvoir arrêter la résolution un peu plus tôt tout en détériorant très peu la valeur de l'objectif, c'est ce que l'on va faire dès que l'objectif n'a pas diminué d'au moins Q pendant 5 itérations successives.

Un autre paramètre que l'on va modifier dans cette section est le maximum de la somme C des compléments des variables sur lesquelles on décide de brancher ($C \geq$

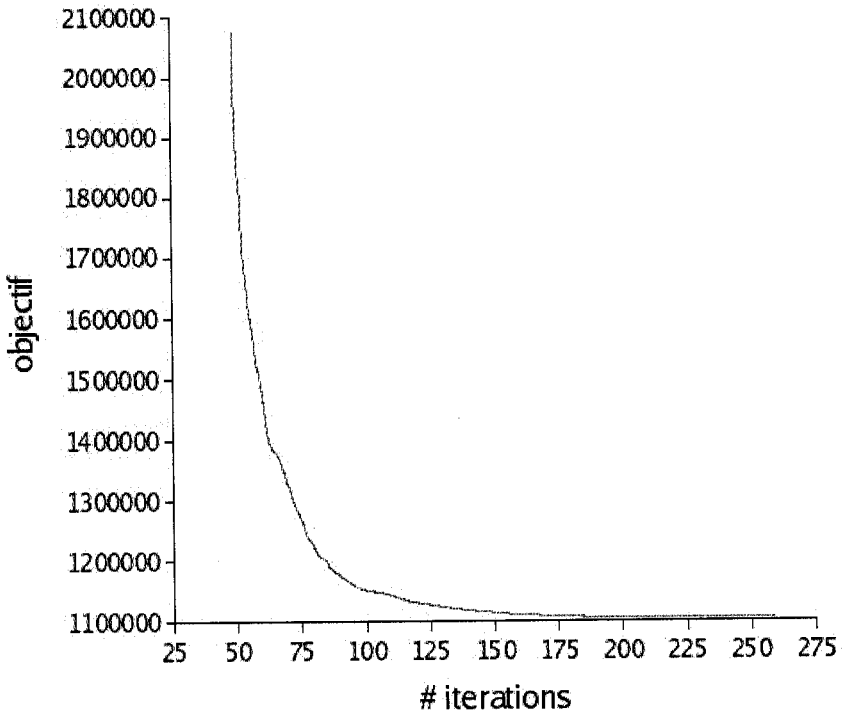

Figure 2.3 – Evolution de l'objectif en fonction de l'itération

$\sum_{kl} 1 - it_{kl}$ avec (k, l) appartenant à l'ensemble des variables d'inter-tâches choisies ou $C \geq \sum_p 1 - \theta_p$ avec p appartenant à l'ensemble des tournées choisies selon le cas). On peut ainsi en le diminuant rendre les conditions de branchement plus sévères.

De plus, on autorise les branchements seulement sur les variables qui ont un flot supérieur à 0.7. Pour les problèmes de taille moyenne, on branchera sur les colonnes et, pour les gros problèmes, on branchera sur les inter-tâches. On conserve ainsi les stratégies les plus intéressantes trouvées précédemment. Les résultats sont rapportés dans les tableaux 2.9 (problèmes de 500 tâches et 4 dépôts) et 2.10 (problèmes de 1000 tâches et 4 dépôts).

Les lignes où $Q = -1$ correspondent à des configurations dans lesquelles on n'arrête pas prématurément la résolution.

Comme prévu, le fait d'arrêter prématurément la méthode de génération de colonnes accélère de façon significative la résolution. Cela est particulièrement vrai pour les gros problèmes (4 dépôts et 1000 tâches). Pour les problèmes de 500 tâches, si on choisit $Q = 500$, le temps de résolution est réduit en moyenne de 13%. Quant aux problèmes de 1000 tâches, si on choisit $Q = 2000$, on observe un gain moyen de 38%.

Mais, les sauts augmentent lorsque l'on arrête prématurément la résolution. Il faut donc pour choisir une stratégie savoir de combien de temps on dispose. Si, par exemple, on impose que la moyenne des temps de résolution pour les petits problèmes (500 tâches) doit être inférieure à 400 secondes, on en déduira que la stratégie la plus efficace est $Q = 50$ et $C = 1$. Pour les gros problèmes (1000 tâches), si on impose une limite de 4000 secondes, on choisit la stratégie $Q = 500$ et $C = 4$.

L'évolution en fonction du paramètre C est plus difficile à analyser. On peut néanmoins remarquer que, dans la plupart des cas, lorsque C augmente le temps de

Tableau 2.9 – Arrêt prématuré : 4 dépôts et 500 tâches

Q	C	saut(%)	temps(s)	noeuds
50	1	0.021	393	13.6
50	2	0.038	386	12.4
50	4	0.023	403	8.2
50	∞	0.022	380	8.6
100	1	0.032	394	17.2
100	2	0.042	387	14.4
100	4	0.052	380	11.6
100	∞	0.026	380	6.4
250	1	0.053	379	16.4
250	2	0.064	373	15.8
250	4	0.083	367	11.8
250	∞	0.073	372	12.2
500	1	0.073	383	24.4
500	2	0.086	365	16.4
500	4	0.092	365	13.4
500	∞	0.105	366	14.4
-1	∞	0.013	427	8

résolution diminue, ce qui est logique car on prend alors plus de décisions de branchement simultanément. Selon ce principe, les sauts devraient également augmenter avec C ce qui est le cas pour les instances avec 1000 tâches mais pas tout à fait pour les instances de 500 tâches. Dans les tests, on observe que les économies de temps ne sont pas très importantes et donc, dans la plupart des cas, ne justifient pas la sélection de paramètres donnant des temps de résolution plus rapides au détriment de la qualité des solutions.

Tableau 2.10 – Arrêt prématuré : 4 dépôts et 1000 tâches

Q	C	saut(%)	temps(s)	noeuds
100	4	0.018	4392	18.2
100	8	0.018	4281	8.8
100	∞	0.026	4274	4.4
200	4	0.022	4230	14.6
200	8	0.031	4055	12.4
200	∞	0.034	4010	3.6
500	4	0.030	3909	15.4
500	8	0.056	3867	12
500	∞	0.074	3908	6.8
1000	4	0.042	3875	15.4
1000	8	0.088	3694	11.6
1000	∞	0.130	3678	7.8
1500	4	0.070	3676	17.4
1500	8	0.150	3576	14.2
1500	∞	0.224	3576	7.2
2000	4	0.168	3470	16
2000	8	0.317	3331	12.6
2000	∞	0.479	3285	8
-1	4	0.0134	6479	17
-1	8	0.0144	5891	11.6
-1	∞	0.009	5424	3.6

2.4 Réoptimisation partielle

Dans cette section, nous avons essayé de guider la résolution par la méthode de génération de colonnes en utilisant une bonne solution précédemment trouvée. On explique la méthode dans la première sous-section et on présente les résultats dans la deuxième. Cette méthode est inspirée du branchement local introduit par Fischetti et Lodi (2003).

2.4.1 Description de la méthode

A partir d'une solution réalisable (dans notre cas, une solution calculée par la méthode de génération de colonnes présentée ci-haut), nous proposons d'effectuer une réoptimisation partielle de cette solution. Toutefois, au lieu de fixer un certain nombre de tournées spécifiques de cette solution, une contrainte exigeant qu'un certain nombre de tournées de cette solution soit conservé est ajoutée au modèle. Cette contrainte laisse donc le choix quant aux tournées à conserver. Nous espérons ainsi obtenir des solutions de meilleure qualité.

On note α la proportion de tournées conservées ($\alpha \in [0,1]$) et Ω^s l'ensemble des indices p correspondant aux tournées de la solution initiale.

La nouvelle formulation du problème est alors :

$$\text{Minimiser} \qquad \sum_{p\in\Omega} c_p\theta_p \tag{2.16}$$

sujet à

$$\sum_{p\in\Omega} a_{ip}\theta_p = 1 \qquad \forall i \in N, \tag{2.17}$$

$$\sum_{p\in\Omega} b_p^k\theta_p \leq v^k \qquad \forall k \in M, \tag{2.18}$$

$$\sum_{p\in\Omega^s} \theta_p \geq \alpha \mid \Omega^s \mid \tag{2.19}$$

$$\theta_p \in \{0,1\} \qquad \forall p \in \Omega. \tag{2.20}$$

Le schéma global de la méthode est présenté à la figure 2.4.

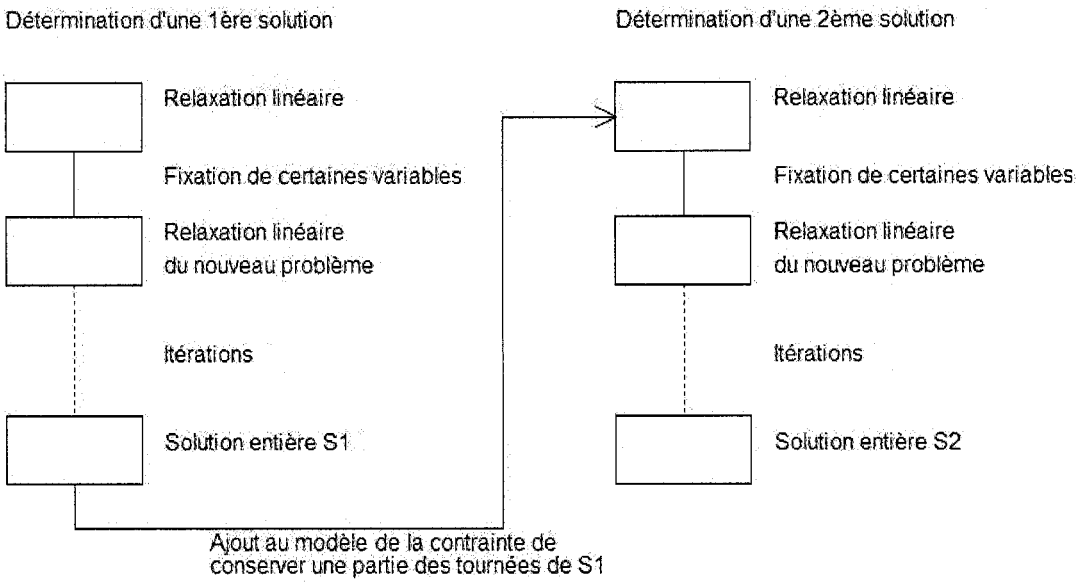

Figure 2.4 – Réoptimisation partielle

L'implantation de cette méthodologie dans GENCOL étant assez complexe, nous avons choisi de séparer le processus en deux étapes bien distinctes. La seule information transmise de la première résolution à la seconde est la solution. On pourrait probablement accélérer la seconde résolution, en recopiant l'ensemble des colonnes présentes dans le problème maître à la fin de la première résolution ou celles présentes à la fin de la relaxation linéaire.

2.4.2 Résultats

Nous avons fait varier le nombre de tournées conservées entre 50% et 90%. Pour chacun des cas, nous avons testé trois stratégies de branchement. Les stratégies de branchement utilisées sont les branchements sur les colonnes supérieures à 0.7, 0.8 ou 0.9 selon le cas. Les solutions initiales utilisées sont celles correspondant aux meilleures stratégies trouvées à la section 2.3 : branchement sur les colonnes plus

grandes ou égales à 0.7 pour les problèmes de 500 tâches et branchement sur les inter-tâches plus grandes ou égales à 0.7 pour les problèmes de 1000 tâches. On rappelle que cela permet d'obtenir un saut moyen de 0.013% en un temps moyen de 427 secondes pour les premiers problèmes et 0.009% en 5424 secondes pour les seconds.

Les tableaux 2.11 et 2.12 présentent, pour les deux groupes de problèmes, la valeur moyenne des sauts obtenus après la deuxième résolution et le temps CPU correspondant uniquement à la réoptimisation partielle. Pour avoir le temps de calcul total, il faut donc ajouter le temps de calcul de la première résolution (i.e. 427s ou 5424s selon le cas). Les moyennes sont faites sur les mêmes cinq instances utilisées précédemment.

On rappelle que les temps de résolution précédemment obtenus pour les stratégies testées ici (07, 08 ,09) sont respectivement 427s, 490s et 530s pour les problèmes de 500 tâches et 5424s, 6600s et 7600s pour les problèmes de 1000 tâches. Les temps obtenus sont donc comparables pour les premiers problèmes, les seconds sont dans le cas de la réoptimisation partielle un peu plus longs (entre 20% et 40%).

Tableau 2.11 – Réoptimisation partielle : 4 dépôts et 500 tâches

	07		08		09	
tournées conservées	saut	temps	saut	temps	saut	temps
50%	0.0128%	453s	0.0133%	503s	0.0132%	550s
60%	0.0129%	415s	0.0125%	453s	0.0127%	509s
70%	0.0111%	416s	0.0111%	450s	0.0133%	532s
80%	0.0116%	492s	0.0116%	510s	0.0119%	602s
90%	0.0129%	410s	0.0125%	462s	0.0127%	541s

On observe que les meilleures solutions sont obtenues lorsque le nombre de tournées conservées n'est ni trop élevé ni trop faible. Dans le deux cas, on arrive à diminuer le

Tableau 2.12 – Réoptimisation partielle : 4 dépôts et 1000 tâches

	07		08		09	
tournées conservées	saut	temps	saut	temps	saut	temps
50%	0.0091%	7941s	0.0091%	8827s	0.0078%	9831s
60%	0.0088%	7789s	0.0083%	8478s	0.0088%	9927s
70%	0.0087%	7443s	0.0084%	8056s	0.0087%	9990s
80%	0.0087%	8128s	0.0075%	8361s	0.0072%	10894s
90%	0.0088%	7914s	0.0083%	8909s	0.0090%	9955s

saut d'environ 20%, ce qui est beaucoup compte tenu de la taille des sauts. Cependant les temps de calculs supplémentaires nécessaires sont assez importants : ils impliquent de doubler voire de tripler le temps de calcul total. Pour les instances de 500 tâches, la stratégie la plus efficace est de conserver 70% de tournées et de brancher sur les inter-tâches ayant des valeurs supérieures à 0.7. On obtient ainsi le plus petit saut en un des temps les plus faibles. Pour les instances de 1000 tâches, la meilleure stratégie dépend du temps dont on dispose pour la résolution.

CHAPITRE 3 : RECHERCHE TABOU

Des méta-heuristiques sont parfois utilisées pour résoudre des problèmes NP-durs. On peut ainsi obtenir des solutions de qualité raisonnable en des temps raisonnables. La recherche tabou fait partie de ces nombreuses méthodes. Elle a été proposée en 1986 par Glover. Dans plusieurs cas, elle s'est avérée efficace en fournissant des solutions proches de l'optimum et se classant ainsi parmi les méthodes les plus efficaces. Ces succès ont rendu cette méthode assez populaire.

Dans ce chapitre, on s'intéressera à la recherche tabou et on tentera de l'appliquer au MDVSP. On commencera par décrire de façon générale la méthode tabou (section 3.1) , puis on décrira l'algorithme développé par Cordeau, Laporte et Mercier (2001) sur lequel on s'est basé (section 3.2). Ensuite, on présentera quelques résultats partiels (section 3.3) qui démontrent que des adaptations sont nécessaires pour rendre l'algorithme de Cordeau et al. plus efficace dans notre problème particulier. Ces adaptations seront présentées dans la section 3.4. Enfin, on analysera l'impact de ces modifications dans la section 3.5.

3.1 Description générale de la méthode tabou

La recherche tabou est une méthode de recherche locale. Elle parcourt donc l'espace des solutions en passant d'une solution à une autre qui appartient au voisinage de la première (i.e. la seconde est obtenue en appliquant à la première une transformation locale simple). Cette méthode autorise les mouvements qui détériorent l'objectif pour se sortir des minima locaux. La particularité de cette méthode est l'utilisation d'une mémoire dynamique appelée liste taboue. La liste taboue contient une liste de mouvements interdits, elle est actualisée à chaque itération. Cette liste permet de restreindre les voisinages explorés et ainsi d'éviter les cycles. En principe, les mouvements sont ajoutés à la liste taboue pour un certain nombre d'itérations. Néanmoins on peut sortir un mouvement de la liste taboue s'il satisfait un critère d'aspiration. Le critère d'aspiration le plus couramment utilisé est d'autoriser un mouvement si celui-ci permet d'atteindre une solution dont la valeur est inférieure à celle de la meilleure solution déjà visitée (dans le cas d'une minimisation). La méthode est fortement dépendante de ce que l'on va choisir comme voisinage et de la façon dont on gère la liste taboue. Un processus de diversification est souvent ajouté pour favoriser les solutions qui ne contiennent pas les caractéristiques les plus fréquemment rencontrées. Formulé autrement, on commence par construire une solution initiale et tant qu'aucun critère d'arrêt n'est satisfait, on se rend à la meilleure solution admissible (non taboue ou autorisée par aspiration) appartenant au voisinage de la solution courante. De plus, à chaque itération, on met à jour la liste taboue et la meilleure solution rencontrée si nécessaire.

La méthode est décrite dans l'algorithme 3.1, avec s la solution courante, s^* la meilleure solution rencontrée, f^* la valeur de l'objectif en s^*, $N(s)$ le voisinage de s et $N'(s)$ le sous-ensemble de $N(s)$ admissible, c'est-à-dire non tabou ou autorisé par aspiration.

Algorithme 3.1 Recherche tabou

Initialisation :
Construire une solution initiale s_0
Poser $s = s_0$
Si s_0 est réalisable, poser $s^* = s_0$ et $f^* = f(s_0)$
Recherche :
Tant qu'aucun critère d'arrêt n'est satisfait faire
- Trouver s_{temp} tel que $s_{temp} = argmin_{s' \in N'(s)}[f(s')]$
- $s = s_{temp}$
- Si s est réalisable et $f(s) < f^*$ alors poser $f^* = f(s)$ et $s^* = s$
- Mettre à jour de la liste taboue

Les critères d'arrêts les plus fréquemment utilisés sont un nombre fixe d'itérations, un temps CPU fixe ou un nombre d'itérations sans amélioration. Le plus souvent, on ajoute un processus de diversification pour éviter que la recherche se concentre sur une partie trop restreinte de l'espace des solutions.

3.2 Description de l'algorithme de Cordeau et al.

La fonction $f(s)$ est égale à $c(s)$ si s est réalisable sinon $f(s) = c(s) + \gamma w(s)$ avec $c(s)$ correspondant au coût de la solution, $w(s)$ au coût total des violations sur les fenêtres de temps (i.e. la somme des retards aux commencements des tâches) et γ un paramètre positif ajusté dynamiquement. Ceci va permettre de défavoriser les violations des fenêtres de temps. Le critère d'arrêt sera un nombre fixe d'itérations.

3.2.1 Construction de la solution initiale

Pour construire la solution initiale (voir Algorithme 3.2), on assigne chaque tâche au dépôt le plus proche de son point de départ. Ensuite, pour chaque dépôt, on trie les tâches selon la position angulaire du départ de la tâche par rapport au dépôt peu importe l'angle de référence. On choisit une tâche au hasard, puis on insère les tâches selon leur position angulaire croissante à partir de cette tâche. On insère les tâches de façon à minimiser l'augmentation du temps de trajet si c'est possible sans dépasser la durée maximale d'une tournée. Sinon on l'insère dans la dernière tournée partant du dépôt. Dans notre problème, il n'y a pas de contrainte de durée sur la longueur d'une tournée. On pourrait donc considérer cette longueur comme infinie mais, pour rendre la méthode d'initialisation plus efficace, on peut déduire des données une valeur plus faible en faisant la différence entre le retour au dépôt le plus tard et le départ du dépôt le plus tôt.

Algorithme 3.2 Construction de la solution initiale

Chaque tâche est assignée au dépôt le plus proche du point de départ.
On note n_k le nombre de clients assignés au dépôt k.
Pour chaque dépôt :
- Créer une liste circulaire des clients selon l'angle depuis le dépot
- Choisir un client j au hasard
- Poser $k = 1$
- Utiliser la suite de clients $j, j+1, ..., n_k, 1, ..., j-1$

Pour chaque client i faire :
- Si l'insertion du client i dans la route k entraîne une violation de la contrainte de durée de la tournée, faire $k = min(k+1, m)$
- Insérer le client i dans la route k de façon à minimiser l'augmentation du temps de trajet

3.2.2 Définition du voisinage

Le voisinage $N(s)$ d'une solution est l'ensemble des solutions que l'on peut construire en déplaçant une tâche i d'une tournée k vers une tournée k' avec $k \neq k'$ dans la

solution s (cf. figure 3.1). Les dépôts D et D' peuvent être les mêmes.

On note i^- la tâche qui précède la tâche i et i^+ celle qui la suit. On insère la tâche i entre les tâches j^- et j^+. Les liaisons en pointillés représentent des chemins parcourant plusieurs clients (ensemble des clients qui précèdent i^- ou j^- ou qui suivent i^+ ou j^+).

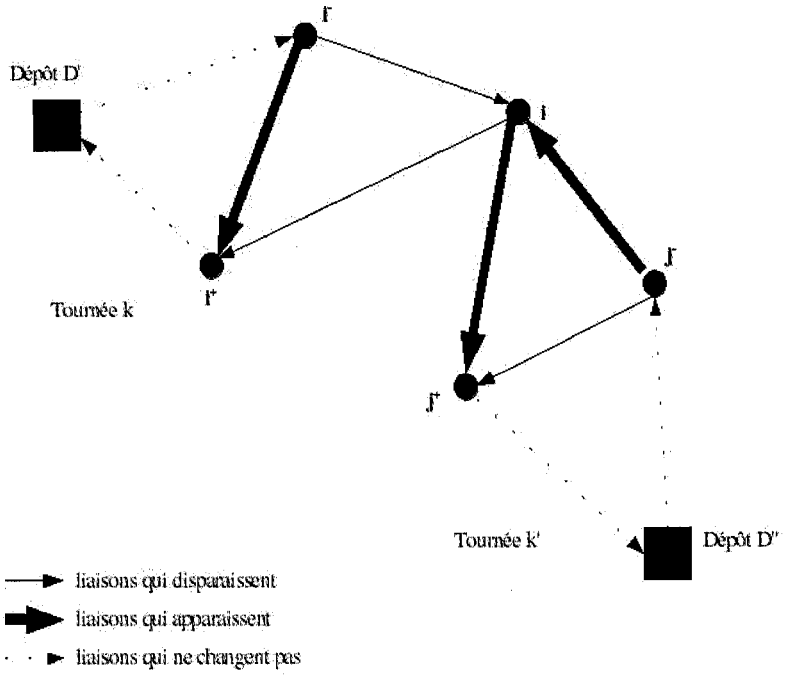

Figure 3.1 – Voisinage (insertion)

3.2.3 Mise à jour de γ

Le paramètre γ permet de favoriser la diminution de l'irréalisabilité de la solution courante. Plus γ est élevé plus l'irréalisabilité diminuera rapidement au détriment du coût de la solution. Au contraire, si γ est faible on va chercher une solution de faible coût sans se préoccuper de l'irréalisabilité et on risque alors de ne jamais rencontrer une solution réalisable. C'est pourquoi on ajuste dynamiquement ce paramètre. On l'augmente lorsque la solution n'est pas réalisable et on le diminue lorsque la solution est réalisable. Plus précisément, si la solution est réalisable (i.e. $w(s) = 0$) alors $\gamma = \gamma/(1+\delta)$ sinon $\gamma = \gamma(1+\delta)$ avec δ un paramètre positif.

3.2.4 Liste taboue

La liste taboue consiste à interdire le déplacement de la tâche i vers la tournée k qu'elle vient de quitter et cela pendant un nombre d'itérations qui suit une loi aléatoire uniforme sur $[0, V]$. V est donc un paramètre qui permet de choisir la longueur moyenne de la liste taboue (qui vaut $V/2$).

On peut révoquer le caractère tabou d'un déplacement de la tâche i vers la tournée k si celui-ci permet d'obtenir une solution meilleure que celle précédemment obtenue où la tâche i appartenait à la tournée k.

3.2.5 Diversification

Pour diversifier la recherche, on ajoute à la fonction f une pénalité p à $t \in N(s)$ lorsque le coût de t est supérieur à celui de la solution courante s. En fait, si $f(t) > f(s)$, on ajoute à $f(t)$ une pénalité $p(t) = \lambda c(t)\sqrt{nm}\sum_{(i,k)} \alpha_{ik}^t \rho_{ik}$ avec ρ_{ik} le nombre

de fois où la tâche i a été affectée au véhicule k, α_{ik}^t un paramètre qui vaut 1 si dans la solution t la tâche i appartient à la tournée k, 0 sinon et λ un paramètre permettant de contrôler l'intensité de la diversification.

3.3 Résultats partiels

Pour obtenir les résultats qui suivent, on a seulement effectué les adaptations nécessaires à la bonne lecture du problème, c'est-à-dire que l'on a modifié la façon dont sont calculés les coûts et les temps des trajets. En effet, dans l'algorithme de Cordeau et al., le coût c_{ij} est égal au temps t_{ij} et il n'est pas arrondi. On a également ajouté la possibilité d'avoir des nombres de véhicules différents à chaque dépôt ce qui n'était pas envisagé.

Dans la figure 3.3 sont représentées les évolutions des résolutions de 5 instances avec 500 tâches et 4 dépôts. On a effectué 20000 itérations en environ 80 minutes sur une station de travail Ultra SPARC II 440Mhz avec 256 MB. Les résultats obtenus ne sont pas satisfaisants car les solutions obtenues sont assez éloignées des relaxations linéaires obtenues au chapitre 2. Elles comportent en fait un nombre trop élevé de véhicules puisque l'algorithme n'a pas été conçu pour minimiser le nombre de véhicules.

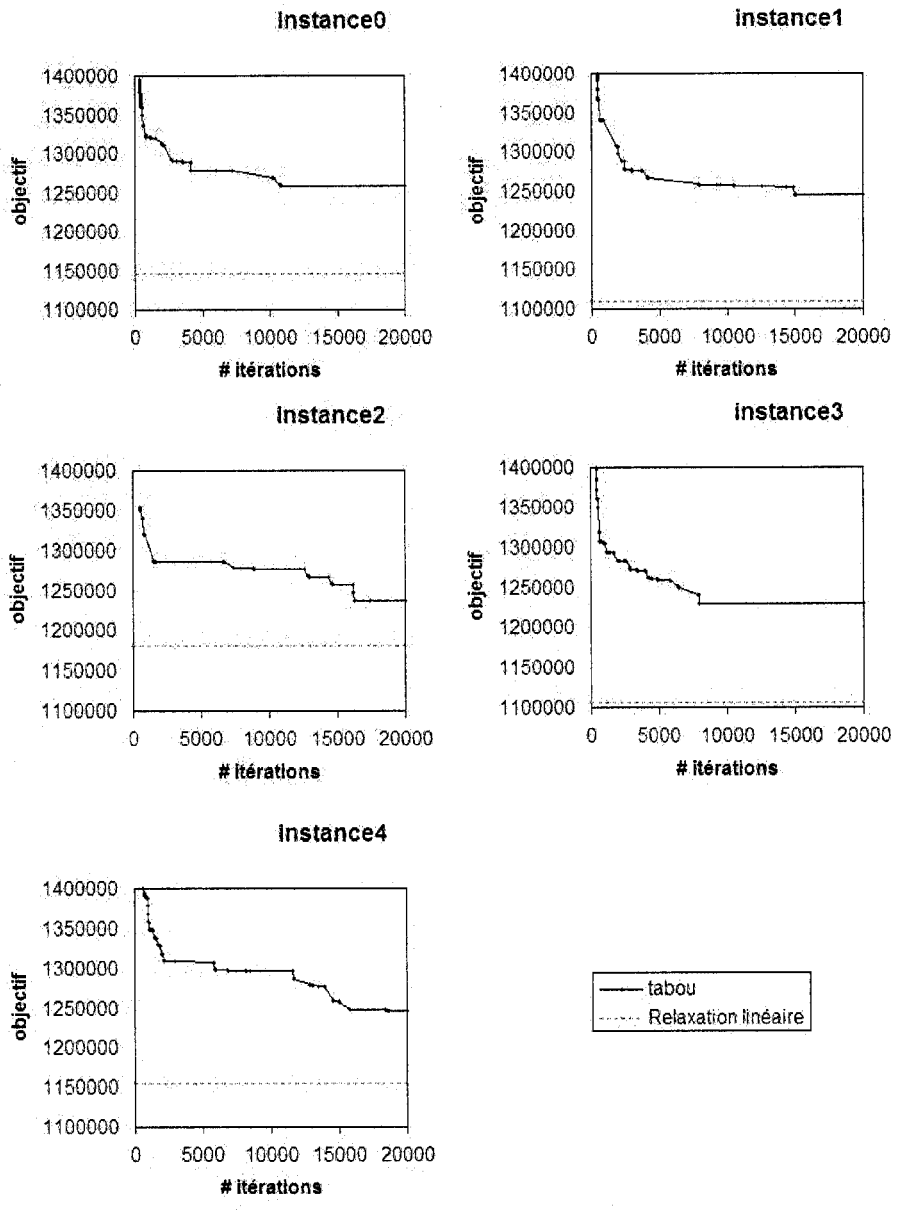

Figure 3.2 – Evolution de l'objectif de 5 instances à 4 dépôts et 500 tâches.

3.4 Adaptations apportées à l'algorithme de Cordeau et al.

On présente dans cette section des essais pour tenter d'améliorer l'algorithme de Cordeau et al. Certains essais se sont montrés bénéfiques, d'autres non. Les adaptations sont présentées dans les sections 3.4.1 à 3.4.5. Une discussion sur ces adaptations est faite à la section 3.4.6.

3.4.1 Autres méthodes pour construire une solution initiale

Nous avons d'abord testé trois autres méthodes pour construire une solution initiale. Celles-ci seront appelées les méthodes d'initialisation 2, 3 et 4, la méthode 1 étant celle présentée à la section 3.2.1.

Méthode d'initialisation 2

La méthode a été proposée par Reimann et al. (2004). Pour construire la solution initiale (voir Algorithme 3.3), on assigne chaque tâche au dépôt le plus proche de son point de départ. Ensuite, pour chaque dépôt, on trie les tâches selon le centre de leur fenêtre de temps (dans notre cas, les fenêtres de temps sont de longueur nulle). On insère les tâches selon l'ordre croissant des centres des fenêtres de temps. On assigne une tâche à chaque véhicule, une fois que l'on a parcouru la liste de véhicules une fois, on la parcourt à nouveau dans le même ordre.

Cette méthode permet d'obtenir une solution presque réalisable sur les fenêtres de temps mais de mauvaise qualité (elle utilise tous les véhicules disponibles). En pratique, les solutions obtenues sont toujours réalisables dans les diverses instances testées.

Algorithme 3.3 Construction d'une solution initiale selon la méthode 2

Chaque tâche est assignée au dépôt le plus proche du point de départ
Pour chaque dépôt :
Trier les clients selon le centre de leur fenêtre de temps
Utiliser la suite de clients $1, 2, ..., n$
- Ajouter chaque client i à la fin de la $j + 1^{me}$ tournée tel que $i = j \ mod \ m$.

Méthode d'initialisation 3

Cette méthode est inspirée de la précédente : on essaie aussi de limiter le nombre de véhicules utilisés et les coûts de déplacement. Pour construire la solution initiale (voir Algorithme 3.4), on trie les tâches selon le centre de leur fenêtre de temps. On insère les tâches selon l'ordre croissant des centres de leur fenêtre de temps. On assigne une tâche à une tournée de sorte que celle-ci reste réalisable. Si c'est impossible, on assigne la tâche à la dernière tournée du dernier dépôt.

Algorithme 3.4 Construction d'une solution initiale selon la méthode 3

Trier les clients selon le centre de leur fenêtre de temps
Utiliser la suite de clients 1,2,...,n
- Assigner chaque client i de telle sorte que la solution reste réalisable tous en minimisant le coût d'insertion
- Si cela est impossible, insérer celui-ci dans la dernière tournée du dernier dépot.

Cette méthode permet d'obtenir une solution réalisable sur les fenêtres de temps si on n'a pas à utiliser tous les véhicules. C'est ce qui passe en pratique.

Méthode d'initialisation 4

Cette méthode consiste à lire une solution résultant de la résolution par GENCOL. Ainsi, on peut tester s'il est possible d'améliorer à l'aide de cette recherche tabou les solutions obtenues par GENCOL.

3.4.2 Voisinage élargi

Le voisinage élargi N(s) d'une solution consiste soit à déplacer un client i d'une tournée k vers une tournée k' avec $k \neq k'$ (cf. figure 3.1), soit à échanger deux clients i et j appartenant à des tournées différentes (cf. figure 3.3). Les dépôts de k et k' peuvent être les mêmes.

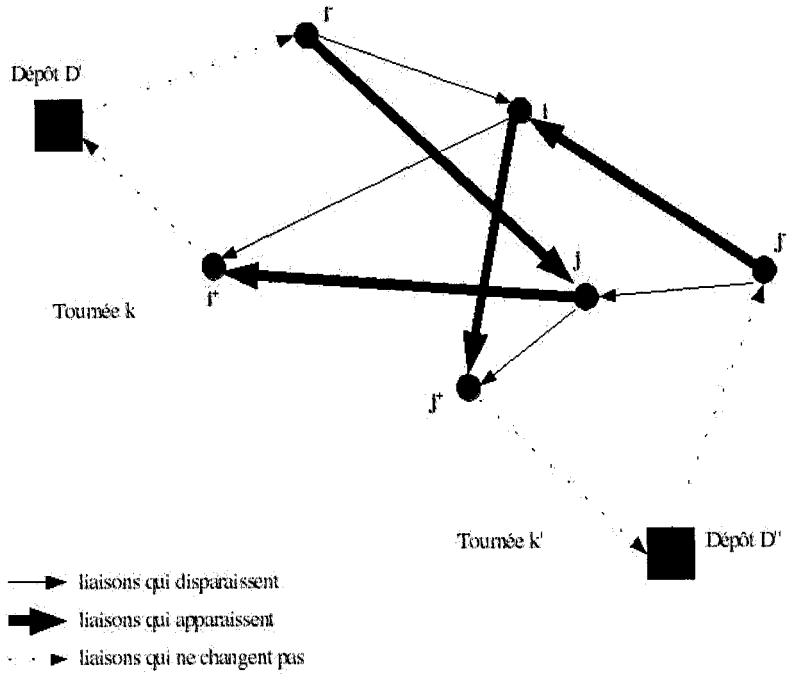

Figure 3.3 – Voisinage (échange)

3.4.3 Modification de la liste taboue

Une autre façon de construire la liste taboue a été testée. Elle consiste à interdire le déplacement d'une tâche i après que l'on ait déplacé celle-ci. Les informations stockées dans la liste taboue sont donc plus contraignantes que précédemment. On espère ainsi éviter des mouvements inutiles. Nous avons également divisé la longueur de la liste taboue en deux paramètres : une constante C et une variable aléatoire V ($\hookrightarrow \mathcal{U}[0, V_{max}]$). La constante C nous assure que le mouvement qui devient tabou le reste pendant un nombre minimum d'itérations ce qui n'était pas le cas précédemment. On pourra révoquer le caractère tabou d'un mouvement à tout moment si celui-ci amène à une solution meilleure que la meilleure solution trouvée jusque-là.

3.4.4 Modifications des mouvements autorisés

On interdit les mouvements de coût nul pour éviter de cycler. Ces mouvements sont assez nombreux car plusieurs tâches ont la même localisation (cf. section 1.4). Pour éviter de rester bloquer sur des solutions irréalisables, toutes les 20 itérations, lorsque la solution est irréalisable, on impose de bouger une tâche qui commence en retard ou la tâche qui la précède.

3.4.5 Modifications pour aider à diminuer le nombre de véhicules utilisés dans la solution courante

L'algorithme de Cordeau et al. n'est pas prévu pour les instances où il y a un coût fixe sur les véhicules. C'est pourquoi il est nécessaire d'envisager des stratégies pour diminuer le nombre de véhicules utilisés.

Diminution de la longueur (en nombre de tâches) des tournées les plus courtes

On attribue des bonus aux mouvements qui vident les tournées effectuant le plus petit nombre de tâches selon le barême du tableau 3.1. On note lmin le nombre de tâches dans la plus petite tournée non vide et P un paramètre permettant de contrôler l'intensité des bonus.

Tableau 3.1 – Barême des bonus

longueur	bonus
lmin	P
lmin+1	P/2
lmin+2	P/3

Suppression de tournées

Après un certain nombre d'itérations avec une solution réalisable, on supprimera une tournée en réaffectant les tâches qui la composent sur d'autres tournées. Lorsque l'on utilise cette stratégie, il faut prendre soin d'interdire les mouvements vers les tournées vides pour éviter que le nombre de véhicules utilisés augmente. On supprime la tournée dont la somme des durées des tâches est minimum. En effet, les tâches de cette tournée devraient être relativement faciles à répartir sur les autres.

3.4.6 Discussions sur les différentes améliorations proposées

Lorsque l'on utilise une seule fois l'algorithme, on utilisera la troisième méthode d'initialisation (section 3.4.1). Nous avons observé dans le cadre de nos expériences que

celle-ci permet d'obtenir une solution réalisable très rapidement. De plus, cette solution est de meilleure qualité que celles obtenues avec les autres méthodes. Néanmoins, comme cette méthode construit toujours la même solution initiale, elle nuit fortement à l'exploration de portions différentes de l'espace des solutions. Elle est donc déconseillée si l'on décide d'utiliser plusieurs fois l'algorithme tabou pour ensuite conserver la meilleure solution obtenue.

L'adoption du voisinage élargi (section 3.4.2) est plus délicate. La décroissance de la valeur de l'objectif est beaucoup plus rapide en nombre d'itérations, mais celles-ci sont beaucoup plus longues. On présente des tests à la section 3.5 qui permettent de faire un choix entre la conservation ou non de ce voisinage.

En ce qui concerne la liste taboue (section 3.4.3), nous avons décidé de conserver sa structure initiale (interdiction de déplacer la tâche i vers une tournée précise) car la modification de celle-ci n'a pas apporté d'amélioration. Par contre, nous conservons le nouveau critère d'aspiration (aspiration si on atteint une nouvelle meilleure solution) qui est beaucoup plus classique. On garde également la possibilité d'ajouter une constante au nombre d'itérations durant lesquelles un mouvement est tabou. Cela permet d'améliorer la qualité de la méthode tabou.

L'interdiction des mouvements de coûts nuls (section 3.4.4) s'est montrée bénéfique. Elle est particulièrement efficace lors de la résolution de petites instances (20 tâches). Sans cela, il fallait parfois un grand nombre d'itérations ($\geq$ 10000) pour atteindre une solution optimale. Avec cette modification, on atteint de façon systématique une solution optimale en un faible nombre d'itérations ($\leq$ 1000).

Les mouvements imposés toutes les 20 itérations si la solution est irréalisable (section 3.4.4) sont assez efficaces dans la recherche d'une solution réalisable. Nous avons donc décidé de conserver cette modification.

Les deux techniques proposées pour diminuer le nombre de véhicules utilisés (section 3.4.5) ont donné des résultats très différents. Le fait de favoriser la diminution du nombre de tâches sur les tournées qui en comporte déjà le moins est efficace pour économiser quelques véhicules. Elle n'est pas assez drastique pour être efficace pour la résolution de nos instances qui sont très contraintes. En effet, elle fonctionne au début et permet d'économiser quelques véhicules mais ne permet pas de s'approcher du nombre minimum de véhicules nécessaires. La méthode qui consiste à vider une tournée quitte à rendre la solution irréalisable temporairement s'est quant à elle révélée plus efficace. On conservera donc cette deuxième méthode.

3.5 Résultats / Analyse de l'impact des adaptations

Les courbes de la figure 3.4, chacune correspondant à une adaptation, montrent l'évolution de la valeur de l'objectif en fonction du temps de résolution pour une instance avec 4 dépôts et 500 tâches. Ces courbes sont des moyennes obtenues à partir de la résolution de 5 instances.

La courbe (1) " tabou sans modification " correspond à l'algorithme de Cordeau et al. avec les modifications minimales décrites à la section 3.3.

Le point (2) " GENCOL(cfix 0,7) " correspond à la solution obtenue avec GENCOL (avec la stratégie qui correspond à brancher sur les colonnes qui ont un flot supérieur à 0.7).

La courbe (3) " tabou sans post-optimisation " correspond à l'algorithme de Cordeau et al. auquel on a enlevé la post-optimisation. En effet, l'algorithme de Cordeau et al. comporte une post-optimisation au sein de chaque route lorsque la solution est

réalisable. Celle-ci est très coûteuse en temps et est inutile dans notre cas du fait que les tâches doivent commencer à des moments très précis. En effet, si on change l'ordre des tâches dans une tournée réalisable celle-ci ne l'est plus. Cette modification est conservée pour toutes les autres courbes.

La courbe (4) " suptour " correspond à la configuration (3) à laquelle on ajoute la technique de suppression de tournées.

La courbe (5) " sans échange " correspond à la configuration dans laquelle on a conservé toutes les modifications qui se sont révélées encourageantes mais en conservant le voisinage initial.

La courbe (6) " tabou avec voisinage élargi " correspond à la configuration dans laquelle on a conservé toutes les modifications qui se sont révélées encourageantes et utilisé le voisinage élargi.

Ces courbes montrent clairement que les adaptations décisives sont la suppression de la post-optimisation pour diminuer le temps de résolution et la suppression de tournées pour diminuer de façon importante la valeur de l'objectif. Les autres modifications conservées ont néanmoins un impact positif mais qui reste relativement faible. Cependant on reste loin des performances obtenues par GENCOL.

Les résultats fournis par la génération de colonne sur les instances de tailles 500 et 1000 sont si bons (moins de 0,01%) qu'il n'est pas étonnant que les méta-heuristiques ne soient pas compétitives pour ces tailles de problèmes. Par contre, pour des instances du MDVSP comportant 10000 tâches, une méta-heuristique s'avérerait probablement bien plus utile.

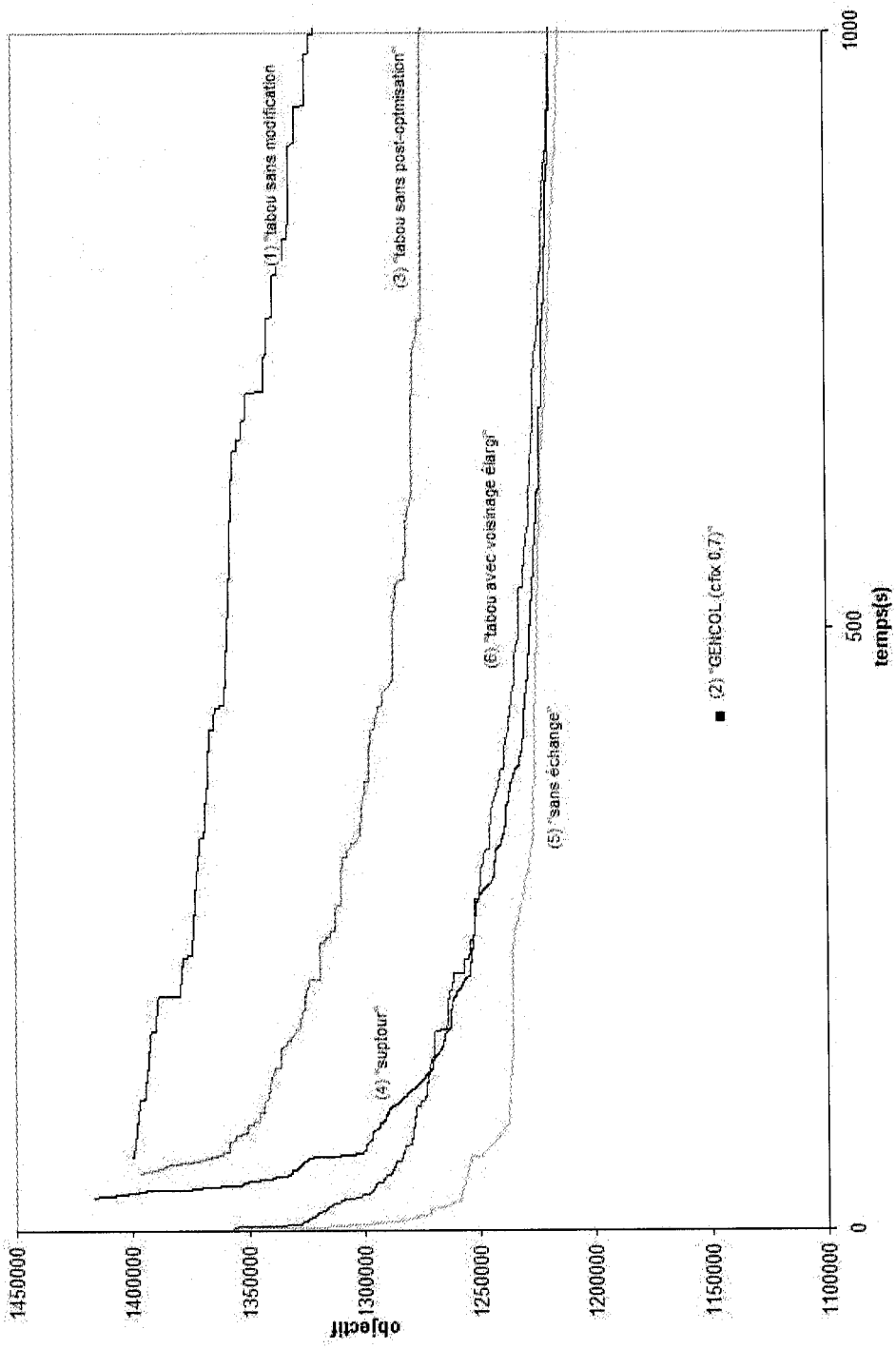

Figure 3.4 – Comparaison des adaptations

Le fait que les tâches doivent être effectuées à un moment précis simplifie fortement la résolution à l'aide de la méthode de génération de colonnes. En effet, les sous-problèmes sont dans ce cas des problèmes de plus court chemin sans ressources sur des graphes acycliques que l'on peut résoudre facilement et très rapidement. Au contraire, cette contrainte pose problème lors de la résolution par une méthode de recherche tabou car en faisant des transformations élémentaires, il est plus difficile d'obtenir des solutions réalisables. La supériorité de la méthode de génération de colonnes sur la recherche tabou pour nos problèmes et les résultats obtenus ne sont donc pas surprenants.

3.6 Post-optimisation des solutions obtenues par GENCOL

La méthode tabou permet de post-optimiser les solutions obtenues avec GENCOL. On a choisi de réoptimiser les solutions obtenues dans les tableaux de la section 2.3.3 car c'est dans ceux-ci que la qualité des solutions est la plus diversifiée. Les résultats sont donnés dans les tableaux 3.2 et 3.3. Dans ces tableaux, seulement la dernière colonne a été ajoutée pour indiquer la diminution du saut suite à la réoptimisation avec la méthode tabou. Le temps de la réoptimisation est de l'ordre de quelques dizaines de secondes.

Tableau 3.2 – Post-optimisation : 4 dépôts et 500 tâches

Q	C	saut(%)	temps(s)	noeuds	dim saut (%)
50	1	0.021	393	13.6	0.00240
50	2	0.038	386	12.4	0.00081
50	4	0.023	403	8.2	0.00093
50	infini	0.022	380	8.6	0.00251
100	1	0.032	394	17.2	0.00133
100	2	0.042	387	14.4	0.00253
100	4	0.052	380	11.6	0.00251
100	infini	0.026	380	6.4	0.00114
250	1	0.053	379	16.4	0.00339
250	2	0.064	373	15.8	0.00490
250	4	0.083	367	11.8	0.00711
250	infini	0.073	372	12.2	0.00474
500	1	0.073	383	24.4	0.00765
500	2	0.086	365	16.4	0.01197
500	4	0.092	365	13.4	0.01380
500	infini	0.105	366	14.4	0.01318
-1	infini	0.013	427	8	0.00000

On observe une petite diminution des valeurs des solutions qui ne coûte pas cher en temps de calcul. Dépendamment de l'importance de cette diminution pour l'usager, la post-optimisation pourrait être utile.

Tableau 3.3 – Post-optimisation : 4 dépôts et 1000 tâches

Q	C	saut(%)	temps(s)	noeuds	dim saut (%)
100	4	0.018	4392	18.2	0.00159
100	8	0.018	4281	8.8	0.00107
100	infini	0.026	4274	4.4	0.00112
200	4	0.022	4230	14.6	0.00153
200	8	0.031	4055	12.4	0.00118
200	infini	0.034	4010	3.6	0.00233
500	4	0.030	3909	15.4	0.00255
500	8	0.056	3867	12	0.00564
500	infini	0.074	3908	6.8	0.00720
1000	4	0.042	3875	15.4	0.00307
1000	8	0.088	3694	11.6	0.00806
1000	infini	0.130	3678	7.8	0.01350
1500	4	0.070	3676	17.4	0.00718
1500	8	0.150	3576	14.2	0.01483
1500	infini	0.224	3576	7.2	0.02289
2000	4	0.168	3470	16	0.01913
2000	8	0.317	3331	12.6	0.03916
2000	infini	0.479	3285	8	0.06799
-1	4	0.0134	6479	17	0.00014
-1	8	0.0144	5891	11.6	0.00005
-1	infini	0.009	5424	3.6	-

CHAPITRE 4 : MÉTA-HEURISTIQUE AVEC UTILISATION D'UNE MÉTHODE DE GÉNÉRATION DE COLONNES À CHAQUE ITÉRATION

Dans ce chapitre, nous proposons une méthode méta-heuristique qui utilise l'efficacité de la méthode de génération de colonnes pour ce problème à chaque itération. À partir d'une solution réalisable, le principe de base est la remise en cause à chaque itération d'un sous-ensemble des tournées. Après avoir choisi les tournées en question, on crée un problème constitué uniquement des tâches appartenant à ces tournées que l'on résout avec la méthode de génération de colonnes. De plus, cette méthode fournira une solution réalisable à chaque itération. Cette méthode est inspirée de la réoptimisation partielle qui a été proposée à la fin du chapitre 2. On essaie néanmoins de limiter la durée de l'opération en fixant les tournées qui sont conservées. Dans ce chapitre, on décrira la méthode ainsi que les résultats obtenus.

Il est à noter que cette méthode possède quelques analogies avec la recherche à voisinage large que propose Shaw (1998). Il résout des problèmes de tournées par un processus de réoptimisation où il remet en cause á chaque itération une partie importante de la solution. Les différences sont néanmoins importantes. Shaw remet en cause des tâches alors que nous remettons en cause des tournées entières. Shaw utilise une seule stratégie pour sélectionner les tâches qu'il remet en cause. Nous en utilisons plusieurs en particulier pour répondre au problème posé par les coûts fixes. Quant à la réoptimisation, Shaw utilise la programmation par contraintes alors que nous utilisons une méthode de génération de colonnes.

4.1 Description de la méthode

4.1.1 Algorithme général

Le principe de base de la méthode consiste à réoptimiser à chaque itération un sous-ensemble de tournées d'une solution réalisable courante. Pour réoptimiser ces tournées, on utilisera une méthode de génération de colonnes heuristique qui nous fournira rapidement une solution de très bonne qualité en autant que le sous-ensemble des tournées est relativement petit. De plus, une liste taboue sera utilisée dans notre algorithme pour éviter de réoptimiser toujours les mêmes tournées. Cette liste sera tout simplement une liste de tournées. Sa gestion est décrite au début de la section 4.1.4.

La solution initiale générée est réalisable. De plus, la méthode de génération de colonnes nous fournit des solutions réalisables pour chaque ensemble de tournées que l'on réoptimise. Ainsi, à chaque itération, la solution courante est réalisable.

La méthode est décrite dans l'algorithme 4.1 où S est la solution courante, f une fonction qui détermine la valeur d'une solution, S^* la meilleure solution rencontrée et f^* la valeur de S^*.

4.1.2 Construction d'une solution initiale réalisable

Pour construire une solution initiale, la dernière méthode introduite au chapitre 3 aurait pu être utilisée. Nous avons essayé d'utiliser une autre approche qui nous semblait avoir un bon potentiel. Cette approche consiste à partitionner les tâches et ensuite à résoudre les sous-problèmes constitués par chaque élément de la partition avec la méthode de génération de colonnes. Dans cet esprit, deux méthodes ont été développées pour construire une solution initiale réalisable.

Algorithme 4.1 Heuristique utilisant la méthode de génération de colonnes

Initialisation :

Construire une solution initiale réalisable S_0

Poser $S = S_0$, $S^* = S_0$ et $f^* = f(S_0)$

Recherche :

Tant qu'aucun critère d'arrêt n'est satisfait faire

- Choisir une stratégie de sélection de tournées
- Appliquer cette stratégie pour trouver les V tournées à réoptimiser
- Utiliser GENCOL pour réoptimiser les V tournées sélectionnées
- Mettre à jour S
- Si $f(S) < f^*$ alors poser $f^* = f(S)$ et $S^* = S$
- Mettre à jour la liste taboue
- Mettre à jour le poids de la stratégie utilisée

1ère méthode

Dans la première méthode, nous créons un élément de partition par dépôt plus un élément de partition supplémentaire. On affecte les tâches au dépôt le plus proche si celui-ci est significativement plus proche que tous les autres dépôts. À cela, on ajoute un sous-ensemble pour les tâches qui sont presque équidistantes à deux dépôts. Les distances ne sont pas symétriques (en général $d_{ij} \neq d_{ji}$), il faut définir la proximité d'une tâche à un dépôt par $p_{ij} = p_{ji} = min\{d_{ij}, d_{ji}\}$.

L'idéal est d'avoir des partitions équilibrées pour que la résolution des sous-problèmes soit la plus rapide possible. De plus, si le problème comporte un grand nombre de tâches, il est judicieux de diviser les sous-ensembles en plusieurs sous-ensembles plus petits pour conserver un temps de construction de la solution initiale raisonnable.

Pour chaque tâche i, on note le dépôt le plus proche $depot[i]$, la distance à celui-ci $dist1[i]$ et la distance au dépôt suivant le plus proche $dist2[i]$. Soit r_{doute} le rapport des distances aux deux dépôts les plus proches à partir duquel on n'affecte pas les tâches à un dépôt précis. Cela signifie, que si $r_{doute} \cdot dist1[i] > dist2[i]$, alors on n'affecte pas

la tâche i au dépôt le plus proche mais au sous-ensemble des tâches indéterminées. La méthode est décrite dans l'algorithme 4.2.

Chaque problème engendré par un des sous-ensembles est résolu séparément en permettant l'utilisation de tous les dépôts. Ainsi, lorsque l'on résout le problème associé à un dépôt, presque toutes les tournées générées sont issues de ce dépôt mais il n'est pas rare qu'une ou deux des tournées créées partent d'un autre dépôt.

Algorithme 4.2 Construction d'une solution initiale réalisable

Initialisation :
Pour chaque tâche $i \in N$:
- $depot[i] = -1$, $dist1[i] = \infty$ et $dist2[i] = \infty$

Affectation :
Pour chaque tâche $i \in N$:
Pour chaque dépôt $D_j \in D$:
Si $p_{iDj} < dist2[i]$
- Si $p_{iDj} < dist1[i]$ alors $dist2[i] = dist1[i]$, $dist1[i] = p_{iDj}$ et $depot[i] = j$
- Sinon $dist2[i] = p_{iDj}$

Désaffectation si doute :
Pour chaque tâche $i \in N$:
Si $r_{doute} dist1[i] > dist2[i]$ alors $depot[i] = -1$

Résoudre les sous-problèmes engendrés par chaque partition

2ème méthode

Dans ce chapitre, nous utilisons le fait que le temps de la résolution par la méthode de génération de colonnes n'est pas linéaire en fonction du nombre de tâches (le facteur est plus élevé). Dans cet esprit, le but est donc lors de l'initialisation de créer des partitions de tailles comparables. Or, la méthode précédente a pour inconvénient de produire des sous-ensembles de tailles différentes et le paramètre r_{doute} est assez difficile à fixer (sa valeur optimale pour équilibrer la taille des partitions change pour chaque instance si les dépôts ne sont pas situés aux mêmes endroits).

Nous allons donc modifier la façon dont sont construits les sous-ensembles. Les tâches seront partitionnées en sous-ensembles de même taille (des sous-ensembles de 100 tâches). Les tâches sont simplement prises dans un ordre aléatoire et les sous-ensembles sont remplis successivement suivant ce même ordre. La localisation des tâches n'est donc plus prise en compte. Cette méthode est plus systématique que la précédente et elle permet de conserver des temps d'initialisation raisonnables même pour des grands problèmes (économie déjà très perceptible pour les problèmes de 1000 tâches).

4.1.3 Choix de la stratégie

À chaque itération, il faut choisir une stratégie pour sélectionner les tournées à remettre en cause. Chaque stratégie i à un poids $w_i > 0$ et une probabilité $p_i = \frac{w_i}{\sum_i w_i}$ d'être choisie. À la fin de chaque itération, on remet à jour le poids de la stratégie choisie $w_i = \rho w_i + (1 - \rho)(f(S_{prec}) - f(S))$ avec $\rho \in [0, 1[$ (S est la solution courante et S_{prec} est la solution à l'itération précédente). Cette formule est inspirée de celle utilisée par Ropke et Pisinger (2004). Le poids est ainsi une moyenne pondérée des améliorations que la stratégie a permises. Les pondérations sont plus élevées pour les utilisations les plus récentes de la stratégie. Les poids des stratégies évoluent donc au cours de la résolution. Cela permet de remettre assez rapidement à jour l'efficacité de la stratégie.

4.1.4 Stratégies implantées

Quatre stratégies ont été implantées pour choisir les tournées à réoptimiser. Elles sont décrites dans les sections qui suivent. Les deux premières sont là pour tenter de diminuer le nombre de véhicules utilisés. La troisième cherche à réoptimiser locale-

ment certaines tournées. La dernière consiste à choisir aléatoirement les tournées à réoptimiser.

Dans les trois premières stratégies décrites ci-dessous, la procédure permettant de sélectionner la première tournée diffère de celle qui permet de choisir les suivantes. Le choix de la première tournée influe sur le choix des suivantes. C'est pourquoi le choix de la première tournée est si important. Pour éviter de choisir trop souvent la même première tournée, on utilise une liste taboue. À chaque itération, la tournée sélectionnée en premier sera donc taboue pendant 20 itérations. De plus, ce caractère tabou n'est utilisé que pour le choix de la première, autrement dit une tournée taboue pourra être sélectionnée si ce n'est pas en tant que première tournée.

Tournées les moins chargées

Nous cherchons ici à sélectionner les tournées les moins chargées, dans l'espoir de pouvoir réaffecter toutes les tâches à un nombre plus faible de véhicules. La charge d'une tournée est égale à la somme des durées des tâches accomplies. Toutes les tournées choisies seront non vides.

Nous commençons par choisir la tournée i non taboue qui minimise une combinaison linéaire de la durée totale des tâches effectuées $duree_i$ et du nombre nb_sel_i de fois où la tournée a déjà été sélectionnée ($duree_i + \beta \cdot nb\_sel_i$). Les tournées suivantes seront les $V-1$ tournées qui minimiseront une fonction du même type, la valeur de β pouvant être différente (en pratique, on prendra une valeur plus petite pour favoriser le critère de durée).

Tournée la moins chargée

Le but ici est de sélectionner une tournée peu chargée dans le but de la vider dans des tournées proches géographiquement. Nous sélectionnons la première tournée de la même façon que dans la stratégie précédente. Par contre, pour compléter l'ensemble des tournées choisies, nous prenons des tournées provenant du même dépôt et n'ayant pas été sélectionnées trop souvent (i.e. dont les nb_sel_i sont les plus faibles).

Tournées proches

Nous choisissons une tournée i aléatoirement parmi les tournées non taboues. Le but est d'essayer d'optimiser le voisinage de celle-ci. Pour cela, nous cherchons des tournées dont une des tâches est proche géographiquement et temporellement d'une tâche appartenant à la tournée i. La stratégie est détaillée dans l'algorithme 4.3 où μ est un coefficient de pondération.

Algorithme 4.3 Stratégie : "Tournées proches"

Sélection de la 1^{re} tournée :
Choisir aléatoirement une tournée i non vide et non taboue
Sélection des tournées "proches"
Pour chaque tournée j :
$pot_j = Min_{k \in i, l \in j}(c_{kl} + \mu t'_{kl}, c_{lk} + \mu t'_{lk})$
avec $t'_{kl} = t_{kl}$ si la tâche l peut suivre la tâche k, ∞ sinon
Sélectionner les $V - 1$ tournées j avec les plus petites valeurs pot_j.

Tournées sélectionnées aléatoirement

V tournées non vides sont choisies aléatoirement. Ces tournées peuvent éventuellement être taboues.

4.2 Résultats

4.2.1 Comparaison des stratégies

Pour tester l'efficacité de la façon dont on guide la recherche, une version de l'algorithme a été programmé où, à chaque itération, on choisit aléatoirement les tournées. Après avoir constaté la suprématie de l'approche où toutes les stratégies sont en compétition (stratégie aléatoire incluse) sur celle purement aléatoire, on s'est posé la question de savoir si la stratégie aléatoire ne ralentissait pas la résolution lorsque toutes les stratégies sont mises en compétition. Pour tester cette hypothèse, une version de l'algorithme où seules les trois stratégies non aléatoires sont en compétition a été introduite. Cette version avec trois stratégies s'est montrée moins efficace que celle à quatre stratégies. On peut en conclure que la présence de la stratégie aléatoire améliore l'efficacité de l'algorithme. En effet, nos stratégies guident assez efficacement la recherche, mais ne cherchent pas dans toutes les directions. Le maintien de la stratégie aléatoire permet de diversifier la recherche.

Une dernière version de l'algorithme a été testée pour s'assurer de l'utilité de la liste taboue. Dans cette version, la longueur de la liste taboue a tout simplement été fixée à 0. Les résultats obtenus sont en faveur d'une conservation de la liste taboue.

Les résultats d'un test des 4 versions de l'algorithme sur 5 instances avec 500 tâches sont illustrés à la figure 4.1. Les courbes représentent l'évolution de la moyenne des valeurs des solutions obtenues pour les 5 instances en fonction du temps de résolution. Les deux figures sont identiques, seul l'échelle change, mettant ainsi en évidence les écarts sur la deuxième courbe.

Les "marches" indiquent le gain d'un véhicule sur une des instance.

4.2.2 Variation de la taille du voisinage

On a fait varier la taille du voisinage (i.e. la valeur de V qui donne le nombre de tournées sélectionnées à chaque itération). Pour cela, des tests sur 5 instances ont été réalisés en faisant varier V entre 10 et 50. Pour comparer simultanément tous les résultats, une courbe par valeur de V est traçée. On place un point pour chaque itération, l'abscisse étant la moyenne sur les 5 instances des temps CPU et l'ordonnée étant la moyenne des objectifs.

Les résultats pour les instances avec 500 tâches se trouvent à la figure 4.3 et celles avec 1000 tâches sont à la figure 4.4. On place de plus sur chacun des graphes un point GC correspondant aux valeurs des solutions obtenues avec GENCOL.

Comme on pouvait s'y attendre, si un nombre trop faible de tournées à réoptimiser est choisi alors notre algorithme n'est pas très performant car il est incapable de faire diminuer suffisamment le nombre de véhicules. Dans les deux cas (500 et 1000 tâches), réoptimiser une quarantaine de tournées à chaque itération semble un bon choix.

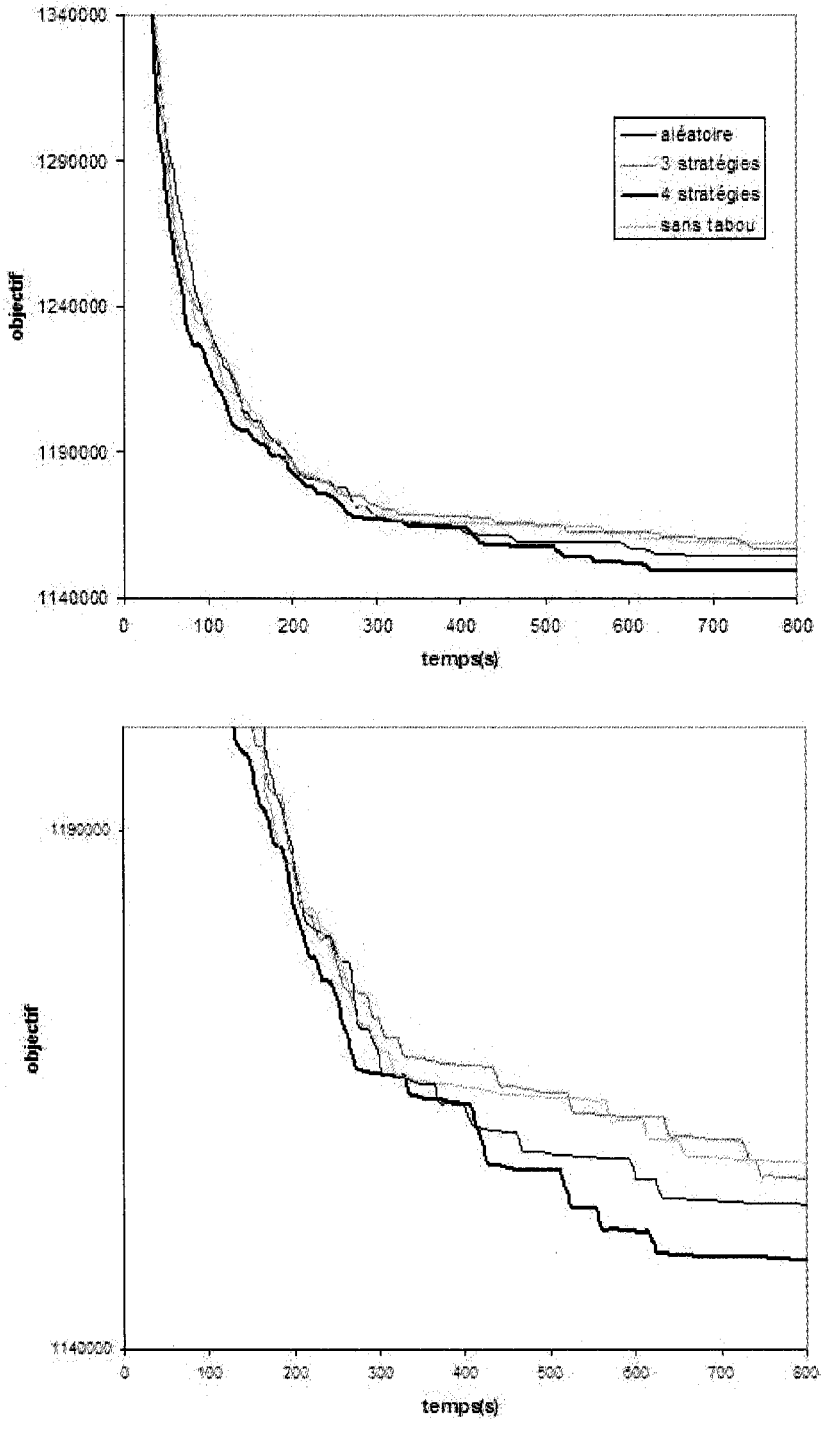

Figure 4.1 – Comparaison des stratégies

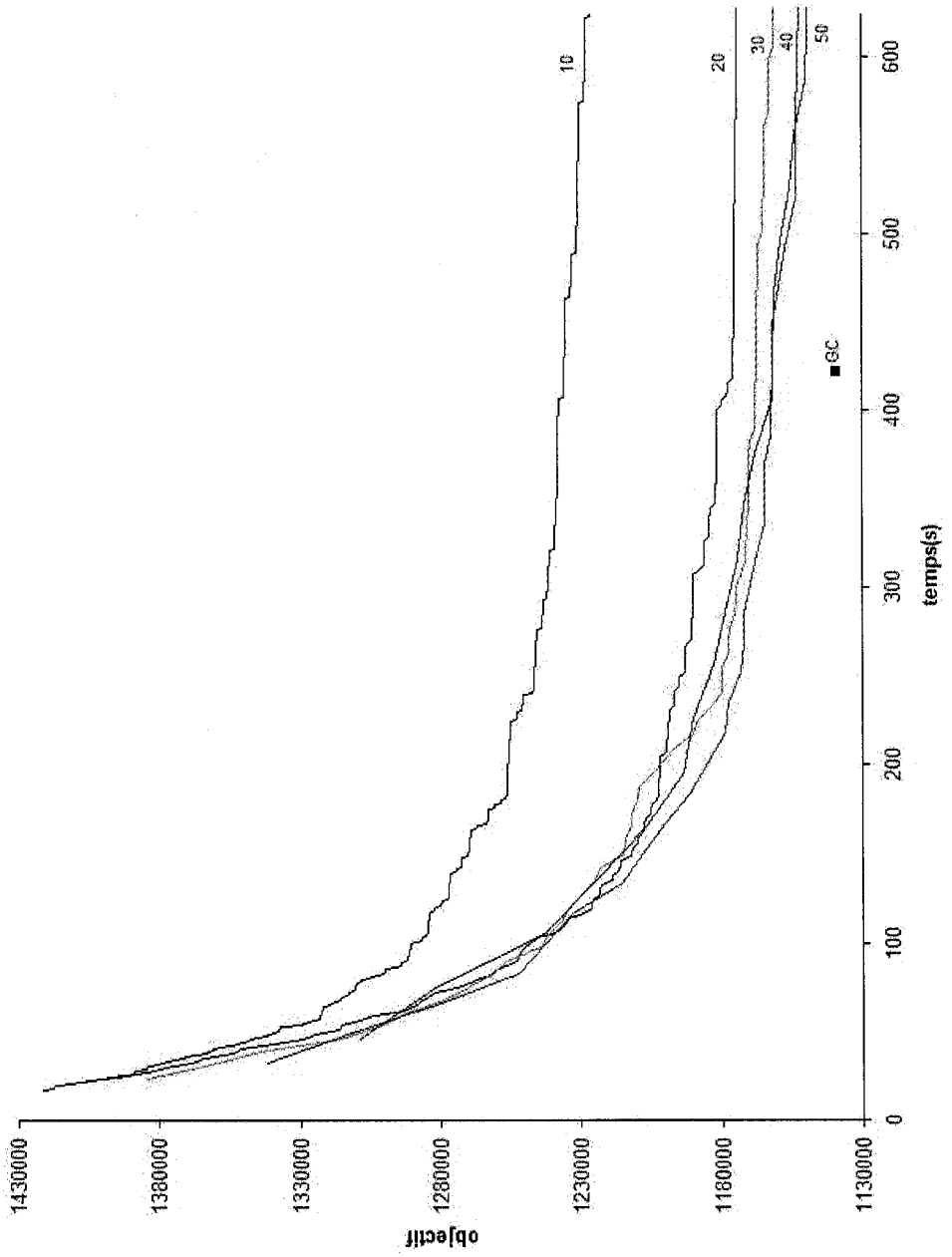

Figure 4.2 – Variation de la taille du voisinage : 4 dépôts et 500 tâches

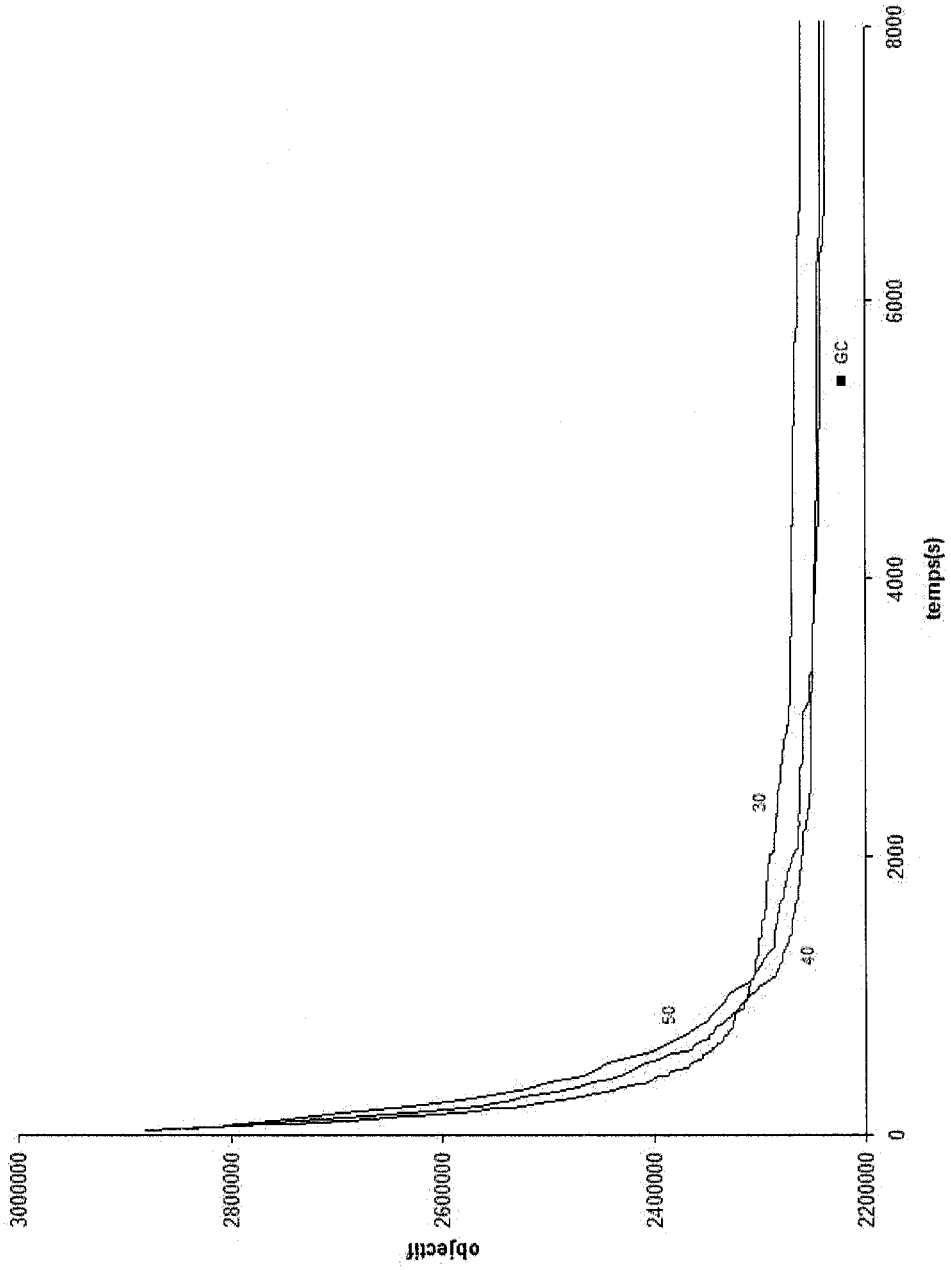

Figure 4.3 – Variation de la taille du voisinage : 4 dépôts et 1000 tâches

4.3 Comparaison des trois approches

Les performances du nouvel algorithme et de la recherche tabou développée au chapitre 3 ont été comparées. Les résultats d'un test sur 5 instances avec 500 tâches sont illustrés à la figure 4.4 et ceux d'un test sur 5 instances avec 1000 tâches le sont à la figure 4.5. Les courbes traçées sont des moyennes sur les 5 instances. On a également ajouté sur ce graphique quelques points correspondant chacun à une résolution par une méthode de génération de colonnes. Les différentes résolutions se distinguent par l'utilisation de différentes valeurs du paramètre Q (voir section 2.3.3) pour obtenir différents temps de résolution.

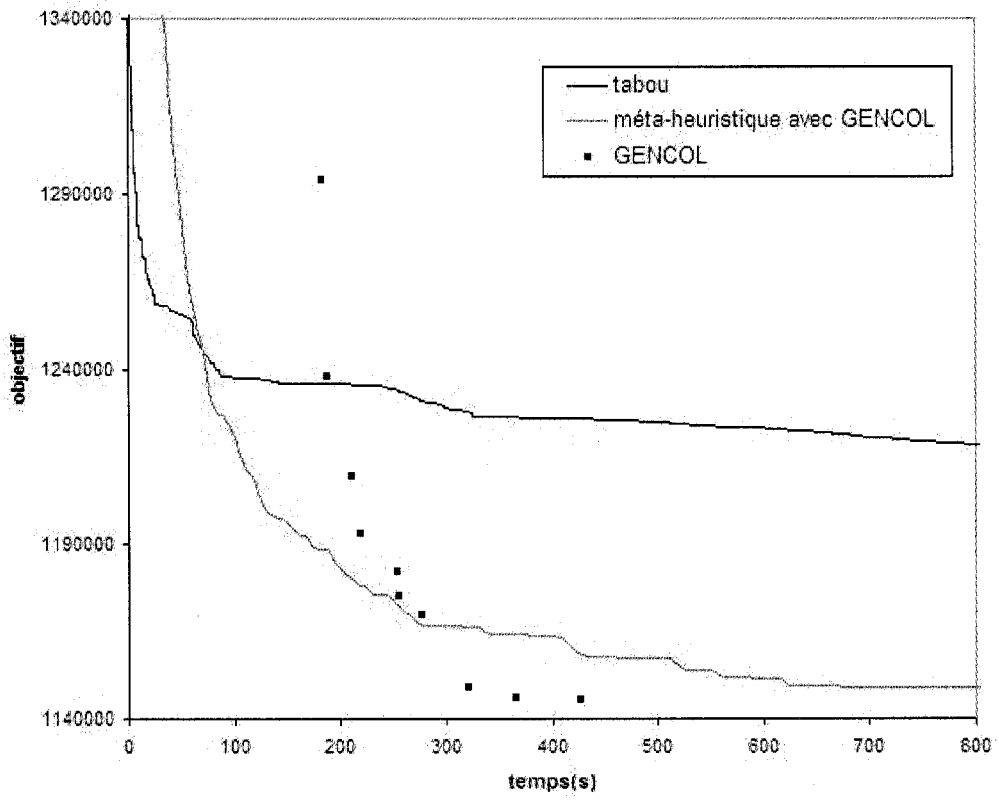

Figure 4.4 – Comparaison des trois approches : 500 tâches

Le nouvel algorithme est plus efficace de façon significative que la recherche tabou, même si la nouvelle façon de construire une solution initiale se révèle moins performante que la dernière introduite au chapitre 3. Le nouvel algorithme surpasse également la méthode génération de colonnes lorsqu'il s'agit de trouver une solution pour les problèmes de 500 tâches en moins de 280 secondes et pour ceux de 1000 tâches lorsque le temps imparti est inférieur à 2900 secondes.

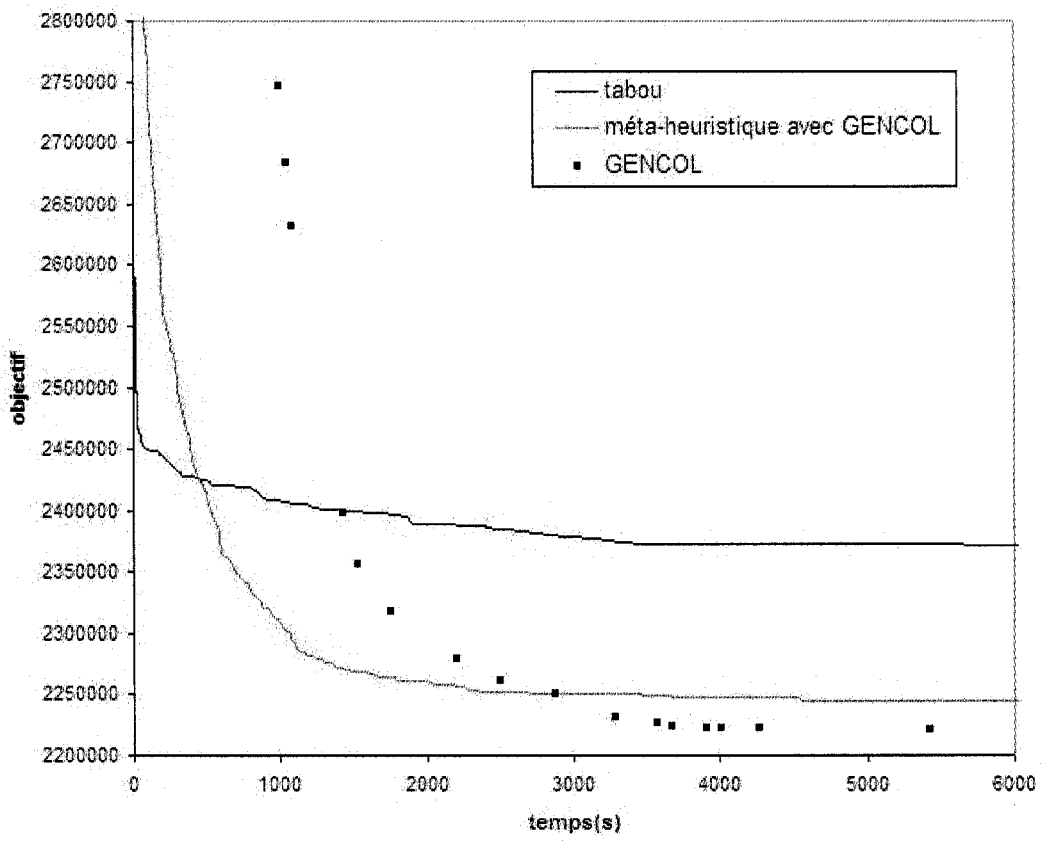

Figure 4.5 – Comparaison des trois approches : 1000 tâches

CONCLUSION

Dans ce mémoire, nous avons comparé une méthode heuristique de génération de colonnes, une méthode de recherche tabou et une méta-heuristique incorporant l'approche de génération de colonnes pour résoudre des instances aléatoires des problèmes d'horaires de véhicule avec dépôts multiples. Les conclusions que nous avons tirées de ce travail sont les suivantes.

Comme le problème étudié est très contraint, la méthode de génération de colonnes est plus efficace que la méthode de recherche tabou pour les tailles des instances étudiées. Les solutions fournies par la génération de colonnes sont très bonnes. En effet, elles ne s'éloignent que d'environ 0,01% des bornes fournies par cette même méthode. La qualité des solutions obtenues avec la recherche tabou est très éloignée de celles obtenues avec une approche de génération de colonnes. Néanmoins, à partir d'une certain taille, la méthode de génération de colonnes éprouve des difficultés alors que la recherche tabou permet toujours d'obtenir des solutions.

Le modèle de réoptimisation partielle introduit à la fin du chapitre 2 s'est révélé décevant car très gourmand en temps de calcul. Il nous a toutefois permis d'envisager l'algorithme du chapitre 4 qui a fourni des solutions de bonne qualité assez rapidement. De plus, pour que cette dernière méthode fonctionne bien, il faut réoptimiser une partie importante des tournées (entre 1/5 et 1/3 de celles-ci).

On peut envisager quelques travaux pour poursuivre ceux effectués dans cette étude. Une première possibilité serait au cours de la méthode de génération de colonnes de produire une solution à chaque noeud de branchement. Pour cela, il faudrait construire une solution à partir des toutes les informations dont on dispose au noeud

de branchement et ensuite réoptimiser celle-ci à l'aide d'une méthode de recherche tabou. Une seconde possibilité serait d'appliquer l'algorithme développé au chapitre 4 à d'autres variantes du problème de tournées de véhicules.

BIBLIOGRAPHIE

BARNHART, C., JOHNSON, E., NEMHAUSER, G., SAVELSBERGH, M., et VANCE, P., (1998). Branch-and-price : Column Generation for Solving Huge Integer Programs. *Operations Research* **46(3)**, 316–329.

BERGER, J., et BARKAOUI, M., (2004). A New Hybrid Genetic Algorithm for the Capacitated Vehicle Routing Problem. *Journal of the Operational Research Society* **54**, 1254–1262.

BERTOSSI, A.A., CARRARESI, P., et GALLO, G., (1987). On Some Matching Problems Arising in Vehicle Scheduling Models. *Networks* **17**, 271–281.

BIANCO, L., MINGOZZI, A., et RICCIARDELLI, S., (1994). A Set Partitioning Approach to the Multiple Depot Vehicle Scheduling Problem. *Optimization Methods and Software* **3**, 163–194.

BIANCO, L., MINGOZZI, A., et RICCIARDELLI, S., (1995). An Exact Algorithm for Combining Vehicle Trips. In : J.R. Daduna, I. Branco, J. Paixão (Eds.), Computer-aided Transit Scheduling, *Lecture Notes in Economics and Mathematical Systems*, Springer, Berlin **430**, 145–172.

CARPANETO, G., DELL'AMICO, M., FISCHETTI, M., et TOTH, P., (1989). A Branch and Bound Algorithm for the Multiple Vehicle Scheduling Problem. *Networks* **19**, 531–548.

CLARKE, G., et WRIGHT, J.W., (1964). Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. *Operations Research* **12**, 568–581.

CORDEAU, J.-F., GENDREAU, M., et LAPORTE, G., (1997). A Tabu Search Heuristic for Periodic and Multi-Depot Vehicle Routing Problems. *Networks* **30**, 105–119.

CORDEAU, J.-F., GENDREAU, M., HERTZ, A., LAPORTE G., et SORMANY, J.-S., (2004). New Heuristics for the Vehicle Routing Problem. Les Cahiers du GERAD G-2004-33, École des Hautes Études Commerciales, Montréal, Canada.

CORDEAU, J.-F., LAPORTE, G., et MERCIER, A., (2001). A Unified Tabu Search Heuristic for Vehicle Routing Problems with Time Windows. *Journal of the Operational Research Society* **52**, 928–936.

DANTZIG, G.-B., et WOLFE, P., (1960). Decomposition Principle for Linear Programs. *Operations Research* **8**, 101–111.

DELL'AMICO, M., FISCHETTI, M., et TOTH, P., (1993). Heuristic Algorithms for the Multiple Depot Vehicle Scheduling Problem. *Management Science* **39**, 115–125.

DESAULNIERS, G., DESROSIERS, J., IOACHIM, I., SALOMON, M., SOUMIS, F., et VILLENEUVE, D., (1998). A Unified Framework for Deterministic Time Constrained Vehicle Routing and Crew Scheduling Problems. In T. Crainic and G. Laporte, editors, *Fleet Management and logistics*, **chapter 3**, 57–93. Kluwer Academic Publisher, Boston.

DESAULNIERS, G., LAVIGNE, J., et SOUMIS, F., (1998). Multi-Depot Vehicle Scheduling Problems With Time Windows and Waiting Costs. *European Journal of Operational Research* **111**, 479–494.

ERGUN, Ö., ORLIN, J.B., et STEELE-FELMAN, A., (2003). Creating Very Large Scale Neighborhoods out of Smaller Ones by Compounding Moves : A Study on the Vehicle Routing Problem. Working paper, Massachusetts Institute of Technology.

FISCHETTI, M., et LODI, A., (2003). Local Branching. *Mathematical Programming B* **98**, 23–47.

FORBES, M.A., HOLT, J.N., et WATTS, A.M., (1994). An Exact Algorithm for Multiple Depot Bus Scheduling. *European Journal of Operational Research* **72**, 115–124.

GILMORE, P.C., et GOMORY, R.W., (1961) A Linear Programming Approach to the Cutting Stock Problem. *Operations Research* **9**, 849–859.

HADJAR, A., MARCOTTE, O., et SOUMIS, F., (2001). A Branch-and-Cut Algorithm for the Multiple Depot Vehicle Scheduling Problem. Les Cahiers du GERAD G-2001-25, École des Hautes Études Commerciales, Montréal, Canada.

LI, F., GOLDEN, B.L., et WASIL, E.A., (2005). Very Large-Scale Vehicle Routing : New Test Problems, Algorithms, and Results. *Computers & Operations Research* **32(5)**, 1165–1180.

LÖBEL, A., (1998). Vehicle Scheduling in Public Transit and Lagrangean Pricing. *Management Science* **44(12)**, 1637–1649.

MESTER, D., et BRÄYSY, O., (2005). Active Guided Evolution Strategies for the Large Scale Vehicle Routing Problems with Time Windows. *Computers & Operations Research*, à paraître.

PRINS, C., (2004). A Simple and Effective Evolutionary Algorithm for the Vehicle Routing Problem. *Computers & Operations Research* **31(12)**, 1985–2002.

REIMANN, M., DOERNER, K., et HARTL, R.F., (2005). D-Ants : Savings Based Ants Divide and Conquer the Vehicle Routing Problem. *Computers & Operations Research*, à paraître.

REIMANN, M., HARL, R., DOERNER, K., et POLACEK, M., (2004). A Variable Neighborhood Search for the Multi Depot Vehicle Routing Problem with Time Windows. Les journées de l'optimisation 2004, Montréal, Canada.

RIBEIRO, C., et SOUMIS, F., (1994). A Column Generation Approach to the Multiple Depot Vehicle Scheduling Problem. *Operations Research* **42**, 41–52.

ROPKE, S., et PISINGER, D., (2004). An Adaptative Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows, rapport de recherche, Université de Copenhague.

SHAW, P., (1998). Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. *Principles and Practice of Constraint Programming, CP98, volume 1520 de LNCS*, 417–431.

TOTH, P., et VIGO, D., (2003). The Granular Tabu Search and its Application to the Vehicle Routing Problem. INFORMS *Journal on Computing* **15**,333–346.